CONTROL ALLOCATION:
THE AUTOMATED DESIGN OF DIGITAL CONTROLLERS

by

Richard J. Cloutier

DRC-01-05-80

April 1980

Control Allocation:

the

Automated Design of Digital Controllers

M.S. Project Report

by

Richard J. Cloutier

Electrical Engineering Department

Carnegie-Mellon University

18 April 1980

This research has been supported by the United States Army Research office, under grants DAAG29-79-C-0197 and DAAG29-78-G-0070, and the Department of Electrical Engineering, Carnegie-Mellon University.

Acknowledgements

I would like to thank my advisor, Dr. Alice Parker for providing me with the opportunity to work on a very interesting project. Her in-depth discussions about the problems involved and her supportive nature have been invaluable. I would also like to thank Andy Nagle for his assistance especially in the representation of the problems inherent to control allocaton.

I would also like to express my thanks to Lou Hafer and Gary Leive who have supplied both detailed information and software support without which this project would have been impossible. In addition, I thank the members of the CMU-DA group who have attentively listened to presentations of my work and have commented on it or have discussed it with me. They have provided me with objective views of my methods, which have pointed out deficiencies that I would never have seen without their help.

Table of Contents

1. Introduction

Digital system design has normally been separated into two main tasks, data path design and control design. During the data path design the architecture of the system takes shape while during the controller design the sequencing and synchronization details are fixed. Control allocation is the process of specifying a controller which will be able to drive the data path design in some specified manner. An automated control allocator will take a description of some digital hardware data path[1] and a procedural description of the desired behavior (the micro-sequence) and produce a description of an engine which will evoke the data path devices in the order specified by the micro-sequence "program". The resulting controller will be a dedicated digital system with its own memory and I/O and have its own specific timing requirements.

Control allocation is a many-faceted problem. Initally the process may be viewed as a hardware design. The functions and interconnection of the digital devices which will generate the sequence of evoke signals must be specified. The control allocation problem is also a problem of autonomous control since the controller must be able to generate all of its own internal control signals which it might require. The system clock signal is an example of such a control signal since it is used to change the state of the controller.

If the controller being designed is of the microprogrammed variety then the control allocation process must also consider the code generation problem. The micro-instructions which will be placed in the micro-rom must first be compiled from the micro-sequence program. To reduce the cost of the controller the control allocator should also consider the problems involved with bit packing of the micro-words. If the micro-word may be reduced in width by a single bit then the microprogram storage requirements will be reduced by as many bits as there are micro-words, a substantial savings in most cases.

Perhaps the most interesting problem of control allocation is the evaluation of potential parallelism. The first aspect of this problem is hardware independence. This is the determination of which basic operations (micro-ops) in the data path machine may be done in parallel due to the independence of the sets of hardware devices which each require. The second aspect is that of data independence in the micro-sequence program. Two micro-sequence steps are data independent when the results of one operation do not depend upon the results of the other.

In addition to these major design problems there are also some intriguing implementation problems to be solved, such as how to correlate different levels of descriptions. If the control sequence is

[1] For a definition of this term and others see section 1. 1

described at a more abstract level than the devices to be controlled are, then each of the control sequence steps must be expanded into a more detailed version which is compatible. There is also the problem of the controller having its own data paths (with the control signals as data) which must be controlled by itself. The question arises: Which comes first the controller data part or the controller control part? The allocator must know the details of the data part before it may define the control signals, but it must know the signal requirements before it may define the data paths fully. In fact, the optimal solution would require simultaneous solutions to both aspects of the problem.

Control allocation is a complete digital design problem, from the basics of hardware interconnections to the details of an optimizing compiler.

1.1 Definition of Terms

Control Points or Control Lines: The inputs of any device which are not defined as data inputs. These lines are able to select a function or evoke an operation in the device. Examples are lines such as clock, load, select, r/w, clear.

Control Signals: The values which must be placed on control points to cause a particular action to be performed. Associated with each device primitive is a control signal for each control line on the device.

Doln Path: A representation of a collection of devices and the interconnections between them.

Data Path Graph: The name for the data path representation in the CMU-DA system. This representation allows for the nodes: register, operator, multiplexer, constant, concatenation, and link. All of the interconnection information is stored in the links and concatenation nodes.

Device: Any collection of hardware elements for which an operation may be defined. Examples are: transistor, AND gate, flip-flop, register, microprocessor. Some device primitives are (respectively): on/off, and, set, load, run/halt/restart.

Device Primitive: The simplest or most basic operation(s) which a device may perform. For example, the simplest type of operation for a register is a LOAD. Whereas the flip-flops which make up the register may be SET but they may not be ON or OFF. The transistors inside of the flip-flop may only be ON or OFF.

Evoke Control Point: A control input which causes a change in the data stored in a device. The clock, load, and clear lines of most devices are in this category.

Micro-controller: The digital subsystem which produces the control signals for driving the data

path devices.

Micro-instruction: The name given to a single word stored in a microprogram rom. It is usually equivalent to the micro-step.

Micro-operation or Micro-op: A collection of device primitives which must be done during a single clock cycle.

The micro-op model is defined as:

micro-operation = <device list,operation time>

An entry in the device list contains:

device list element = (name of device,
source or destination flags,
control signals for this device,
pointer to the next device list element or zero>

operation time = <the time required for this micro-operation to be completed>
(Usually the longest propagation delay from one of the sources to the destination device(s).)

Microprogram Rom: The device (memory) of the micro-controller which contains the micro-instructions. Also called the MCROM or the micro storage memory.

Micro Sequence Table: The part of the CMU-DA design description which contains the micro-sequence program. This table is made up of register transfer instruction and control How directives.

Micro-step: A collection of micro-ops which will be done during the same major clock cycle. Micro-ops are allowed in the same micro step only when there are no device or data conflicts between them.

Register-transfer Operation: An abstract description of the transfer of a value stored in a register to another register through an optional operator. A register transfer may also describe a transformation made upon the data *in situ,* such as a shift in a shift register.

Select Control Point: A control input which does not of itself change the value stored in the device. Function selects and output enable lines are of this type.

1.2 The Basic Requirements

In order to understand how a control allocator works one must first understand what it has to work with. There are three basic requirements and each must be available in order for the allocation

process to operate as an independent process.

The first requirement is a description of each of the devices which will be used to implement the digital system. This description should contain a name for the particular device, the operations which the device may perform, and a concise description of how the particular device may be controlled for each operation. In addition to the devices themselves, the interconnections between them in the digital system is important and must be included in the basic requirements.

The second main requirement for any control allocator is an abstract description of the control sequence. A control sequence is a representation of the proper order of events which should occur in the digital system and may be represented at any level of detail, analogous to the description of the devices. This sequence may be represented in a high level programming language or it may be expressed in minute detail by specifying the actual bit patterns which should be used to drive the control points of the data path devices.

The third and final input requirement for a control allocator is a list of the significant constraints on the design. These may be parameters such as cost and speed or they may be instructions as to which types of controllers should be considered. Each of these restrict the design space and are helpful to the control allocator in its search for a suitable controller.

With these requirements fulfilled the control allocator should be able to define a useful controller.

1.3 The Solution

The micro-sequence program is the heart of the control allocation process. Each instruction will be mapped into one or more basic data path operations (device primitives). Combinations of the device primitives will be used to define micro-operations. Once this has been done for each micro-sequence step then there will exist a micro-operation program at a level of detail sufficient for the completion of the control allocation process. The problem will then be one of determining the form of the controller which should be used. The control allocator will select a particular controller configuration by using parameters measured from the microprogram. Once selected, the controller is entered into the path graph description. The control allocator then evaluates the potential parallelism in the micro-operation program. This is determined from the data and hardware dependencies of successive micro-ops. The micro-operations are then assigned to micro-steps. This assignment is a major problem in itself and the procedure used for this project is a heuristic method which attempts to constrain the micro-word width to a specified value while trying to reduce the execution speed of the micro-controller to a minimum. After the micro-steps are assigned, the size requirements for the micro-storage are known and this information is placed in the data path graph description. The final

step of the control allocation process is the conversion of the micro-operations into bit patterns which will be stored in the microprogram rom. This step is fairly simple and is similar to the assembling of a machine language program.

2. Control Allocation in Relation to Other Design Tasks

Control allocation is only a single step of the complete digital design process. It is not necessarily independent of all of the other steps and in most hand designed projects the controller design is considered during the time that the data paths are being designed. This overlap allows for optimizations in the controller which will reduce the cost or increase the speed of the machine.

2.1 The CMU-DA project

The CMU-DA project is a top-down type of design system. It consists of programs which map the design descriptions from an abstract level to a more detailed level during each step. The inital description of the digital system is in the form of an ISPS procedure. This is converted and modified by successive programs until it is detailed enough for construction. The first step is the Value Trace process. In this step the ISP description is converted into a new language which represents the data flow and control flow of the design in a graphical form. This graph allows the VT program to recognize data and control dependencies which will allow for transforms on the design. Some possible transforms include ones which eliminate unnecessary computations such as the recalculations of values which were previously evaluated and could have been saved. A more advanced transform is similar to the code motion technique used in optimizer compilers. The movement of common operations out of all the branches of select statement and the removal of invariant computations from loops are two such transforms [McFa 78]. The second step is the Design Style Selector which determines which type of design the ISPS description is most similar to, and should be implemented with. The effect of this step is to select which particular data path allocator should be used. The third step, Data Path Allocation, is where the data path devices and their interconnections are selected. This allocator uses generic types of devices as building blocks and assumes that the technology which will be used to implement the design can be used to construct these blocks. The following step is Module Binding, during which the devices in the data path are assigned to specific real devices which are available in the design technology. If a device is not available to perform a requested operation then the module binder transforms the data path graph and the micro-sequence table in such a way that the assumed operation may be performed with available devices. The Control Allocation step is next and here additions are made to the path graph to include the controller hardware. The micro-sequence program which was generated by the Data Path allocator is compiled into a microprogram and the program is added to the description of the design. The final steps of layout and construction take the description and modify it in any way which is required for construction. For a detailed discussion of the CMU-DA system see [McFa 78].

3. The Input Requirements for the CMU-DA Control Allocator

In the CMU-DA project the main requirements for control allocation are satisfied by the combination of three sources. The first is the design description which contains the data path graph and the micro-sequence table. The second is the Module data base which contains control information about all the devices used. The third input is from the user, who selects specific parameters during the control allocation process.

The data path graph contains a list of the devices which are required by the design at its current stage of completion. The data path graph also contains information about how each of these devices are interconnected. Associated with each device is a name of a module which will perform the specified functions. The data path graph has been generated by the Data Path allocator and the module information has been added by the Module Binder. A description of each module is contained in a database called the Module Database. The information concerning how devices may be controlled to perform particular operations is also contained in this database. The combination of the Module Database and the data path graph is sufficient for the first main requirement for control allocation.

Included in the same file as the data path graph is the micro-sequence table. This is the abstract description of the control sequence which is required by the control allocator. The micro-sequence table is a series of register transfer instructions and control flow directives. Each of the register transfer instructions specifically notes which devices of the data path graph should be used for the operation. If the micro-sequence program had not been bound to the path graph in this way then the control allocation problem would have been much more difficult because it would be necessary to associate the program operations with actual hardware devices. This type of conversion and the associated register allocation problems have been considered by DeWitt and Mallett. DeWitt defines the process of register allocation as the procedure necessary to determine which hardware register should be assigned to contain a program variable and when this assignment should be changed to accommodate a new program variable [DeWi 78]. He also deals with the problem of processor allocation used to determine which operator should be used for a particular instruction. DeWitt shows that this problem is NP complete [DeWi 76]. Mallett has also considered the problem of micro-word compaction and states:

> "A high-level language to microcode translator cannot afford the time to exhaustively improve the object code for every moderately sized program"

Mallett also presents a heuristic method which seems to compact nearly optimally for a linear segment of microcode, in a predefined digital system [Mall 78]. These heuristic methods include ones which

direct a search through a branch and bound type graph and a method of early termination of search down branches of this graph. For the current control allocation project however the controller has not yet been defined and Mallett's procedure may not be used.

The micro-sequence program also includes the control directives in the form of ifs, selects, and joins. Each of these only evoke devices in the controller and the specific action of each will be known by the control allocator so there is no register allocation problem here. Thus the micro-sequence table is sufficient to fulfill the second major requirement of control allocation.

The constraints for the design, which is the third major requirement of control allocation are either built into the control allocator itself or they are specified by the user when running the allocator. While there are multiple controller schemes such as asynchronous operation and distributed or residual control, only the microprogrammed variety is considered in this project. The bitwidth of the micro-word, which is related to the cost of the controller, is specified by the user at runtime. The speed of the controller is inversely related to this bitwidth. This effect is due to the fact that wide words are able to control more data path devices at once and thus more concurrent operations are allowed.

4. How Control Information is Stored in the Module Database

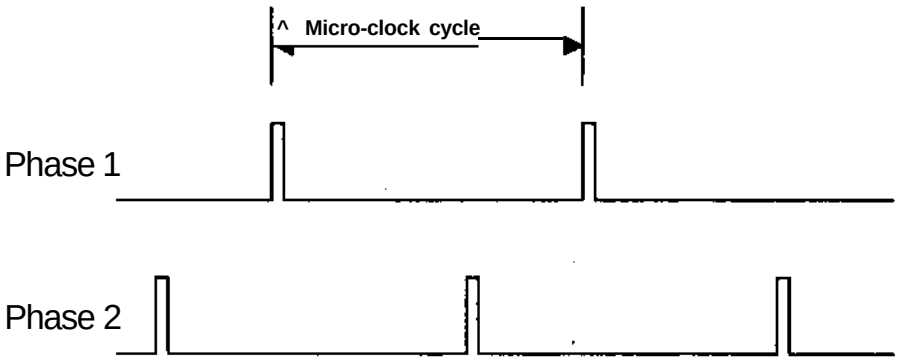
The control information which is being considered here and that which has been placed in the database is only a subset of that found in an ordinary data book. It is only concerned with the operations which will be requested by the register transfer operations and in addition the information has been conditioned by the type of control structure which will be used by the controller.

4.1 Assumptions Made About the Controller

There have been several assumptions made about the operation of the controller before any of the control data was entered in the database. These assumptions were made for two reasons: to save processing time during the control allocation step and to simplify the process of defining the control signals. The controller model which is used assumes that a sequence of signals will be generated. The select signals will become valid first and then the evoke signals will occur. After the evoke signals have returned to their inital state the select signals will become invalid. Allowing for only one type of sequence simplifies the control allocator's job. If arbitrary types of sequences were allowed then the control allocator would have to map each type into the sequence which the controller could generate. Since this mapping would have to occur once for each bit stored in the microprogram it was decided that the person defining the conliol signals would store the standard sequence in the database and save a substantial amount of processing time. With only a standard sequence allowed the process of entering the information in the database should also be simpler for the user.

The first assumption made about the controller is that there will be a two phase clock system in the controller. The general form of these signals is shown in Figure 4-1. The pulse width of the clock signals will be narrow enough so that both the rising and falling edges may be used for evoking actions. The clock cycle time and their relative phase will be determined by the control allocator during the generation of the micro-code.

A second assumption made is that each control point in the database will be classified as either a select or evoke line. The signals which drive select inputs will be wired directly from the micro-controller and their values will be held at the value specified in the data base for a complete clock cycle. Control points which are designated as evoke points will have their control signals conditioned by the phase 2 clock signal. This allows for four types of evoke signals: rising edge, falling edge, positive pulse, and negative pulse. To do this conditioning, the signal from the microprogram word will either be NANDed or ANDed with phase 2 of the system clock. The micro-word value will be determined by the control allocator so that the proper evoke signal will be



The select signals are valid from the start of a phase 1 cycle to the end of that phase. The evoke signals occur during the phase 2 clock pulse.

Figure 4-1: The micro-clock signals

generated. The use of either NAND or AND will also be determined by the control allocator by examining the non-active state specified in the database for the evoke control point. After the operation is evoked the value at the control point will return to its non-evoke value and then the select control lines will be set to the value required for the next operation.

4.2 An Example

For a more detailed explanation of what the database contains consider this example of one of the entries. An partial listing of the SN74161 entry is:

```
II): SN74161
LINE -14-          -CTILINES-      -C1LNAME-
                    5               tCTLNAMF.
    ID: CTI NAME.1
    LINE  -0-      -P1NNAME-       -PINTYPL-       -NONEVOKE-      -SUBMODNO-
                    CIOCK           1               H               1
    ID: CILNAME.2
    LINi:  -0-     -PINNAME-       -PINTYPI-       -NONLVOKE-      -SUBMODNO-
                    CLEAR           1               H               1
    ID: CILNAME.3
    L1NL  -0-      -PINNAME-       -PINIYIH-       -NONLVOKE-      -SUBMODNO-
                    LOAD            1               H               1
    11): CTINAME.4
    UNI   -0-      -IMNNAML-       -IMNIYPI -      -NONLVOKI-      -SUBMODNO-
                    ENBP            0               X               1
    ID: CUNAME.5
    LINE  -0-      -PINNAME-       -PINTYPE-       -NONEVOKE-      -SUBMODNO-
                    ENBT            0               X               1
IINE -14-                                       -CIIISEO-
                                                  tCHESEQ
    ID: INC
    LINE  -0-      -EVOKLINE-      -EVOKSTEP-      -SUBMOD-        -MAXTIME-       -ESEQ-
                    1               1               1               35              tESEO
        ID: ESEQ.I
        LINE  -0-      -EVAL-
                        tEVAL
            II): IVAI .1
            LINE  -0-      -IiI IVAL-
                            P
            II): fVAL.2
            LINE  -0-      -BITVAL-
                            II
            II): IVAI .3
            1 INI  -0      -HI IVAI -
                            H
            ID: EVAL.4
            LINE  -0-      -BITVAL-
                            II
            ID: EVAI.5
            LINE  -0-      -UI IVAL-
                            H


    ID: I0AD
    LINI  -0-      -IVQKI 1NL-     -1VOKSIIP-      -SUHMOD-        -MAX1IML-       -ESEO-
                    1               1               1               29              tESEQ
        11): ESIQ.I
        1 INI  -0-     -IVAL-
                        tEVAL
            ID: IVAI .1
            LINL  -0-      -HI IVAI -
                            P
            ID: IVAI .2
            1 INI  -0-     -III IVAI -
                            H
            ID: IVAI.3
            LINE  -0-      -HI IVAL-
                            L
            ID: IVAI .4
```

```
        LINE  -0-       -BITVAL-
                         H
        ID: EVAL.5
        LINE  -0-       -BITVAL-
                         H

ID: READ
LINE  -0-       -LVOKLINE-    -tVOKSIEP-    -SUBMOD-      -MAXTIME-     -ESEQ-
                 0             0             1             0             tESEQ

ID: CLEAR
LlNl  -0-       -LVOKLlNr-    -LVOKS1LI'-   -SUBMOD-      -MAX!IME-     -ESEQ-
                 2             1             1             38            tESEQ
    ID: LSEQ.l
    LlNl  -0-       -EVAL-
                     tEVAL
        ID: EVAL.l
        LINE  -0-       -BIIVAL-
                         H
        ID: IVAI.2
        LINE  -0-       -BITVAL-
                         L
        ID: EVAL.3
        LINE  -0-       -BITVAL-
                         H
        1D: EVAL.4
        LINE  -0-       -BITVAL-
                         H
        ID: EVAL.5
        LINE  -0-       -BIIVAL-
                         H
```

In this example only line 14 is shown, however in the data base there are lines 0 through 13 which contain other information about this particular device. See Leive's report on the module database for more information about the lines O through 13. [Leiv 79]

Line 14 contains the information required to specify control for the device. The first entry, CTLLINES, is the number of control lines for each device, in this case it is five. Note that this is the number of lines per device and not the number of lines per package. In the SN7474 there are two devices per package with six control lines total but only three CTLLINES per device. If CTLLINES is zero then there are no control lines for the device and it requires no control inputs to perform its function. The SN7400 (NAND) is such a device.

The second entry CTLNAME is a list of characteristics of each control line. There are four traits for each line and the first, PINNAME, is a character string which names the control point on the device. The second trait, PINTYPE, is either 1 or 0.. A 1 signifies that the particular control point is an evoke input, a 0 indicates it is a select input. The third trait is NONEVOKE which is a single character indicating the non-evoke state for a particular line. The three valid values are H(high) L(low) and X(don't care). The NONEVOKE has an obvious meaning for evoke lines but if a particular device requires some sort of setup sequence then the value of NONEVOKE for a select line could be

something other than the expected X. The final trait SUBMODNO is the submodule number of the control point. A submodule is defined as the set of control points which are required to cause a particular operation to be performed.

The third section of line 14 is a list of the operations which may be performed with the particular module. This has the heading of CTLESEQ. In the above example the module may perform the operations: INC,LOAD,READ,and CLEAR. The INC,LOAD and CLEAR should be obvious as to what the particular functions is. The READ is included so that any micro-operation which requires this module as an input will be able to find out if any control lines need to be set in order to read the contents of the register. In this case there are no output enable lines so there is no ESEQ (Evoke SEQuence) for a READ. Some other devices might have such a line which must be controlled. To further explain the CTLESEQ consider the INC operation. The EVOKLINE is a number which indicates which of the EVALs (Evoke VALues) is the evoke control point for this particular operation. The evokstep is the number of the ESEQ during which the operation is performed. If a particular operation requires either an intricate setup or hold sequence on the control lines then it may require more than a single ESEQ (control step) and EVOKSTEP should indicate which step the evoke is actually performed. One such example might be the multiplication of two numbers by a special function unit which requires the following steps: Load num1 in register A, Load num2 in register B, Start the multiplication, Get the result. Such an example would have an ID of MULT and would require four ESEQ steps.

The SUBMOD is the submodule number which is used by this operation. If there is more than one submodule in this device then for the current operation (INC) there would be fewer control values listed than the number of control lines for the device. MAXTIME is the time in nanoseconds required for this module to perform the current operation.

ESEQ is a list of the steps which must be followed to perform the operation. If more than one step is required for the operation then there is more than one ESEQ. Each ESEQ.n has a list of the EVALs which the control points must be set to. There are as many entries in the EVAL list as there are control lines in the submodule being used. The EVALs are listed in the same order as the CTLNAMEs list but only the current submodule control points are included. If the line named LOAD of the above example had been in submodule 2 then for the INC operation there would have been only four EVALs listed. EVAL.1 would correspond to CTLNAME.1, and EVAL.2 with CTLNAME.2, EVAL.3 with CTLNAME.4, EVAL.4 with CTLNAME.5. There would be no EVAL.5.

The BITVALUE is a character which indicates what type of control values should be used on the particular control point. Possible values for BITVALUE are P(rising edge), N(falling edge), H(positive pulse), L(negative pulse), S(select), X(don't care). The select is used for the select bits on a

multiplexer and the actual value stored in the microprogram is determined by the control allocator from the path graph link selected. One of the advantages of this control information structure is that it is a simple matter to determine the value of the bits to be stored in the micro-word.

The assumed model of the controller allows the user to squeeze some operations, which would have required more than a single cycle in a simpler controller, into a single step (ESEQ). Note that in this example the 74161 requires that the two count enable lines (ENBP and ENBT) be changed only while the clock line is high. With the NONEVOKE value of the clock equal to H the select lines may be changed to any value before the clock goes low then high (the rising edge is the evoke signal) to perform the selected operation. If the controller could not have set the non-evoke value of the clock line then this would have required two control steps.

5. The Steps of Control Allocation

The first main step that the control allocator does is to read the interconnection information from the path graph and places it in an internal form which will allow for tracing through the data paths in search of controllable devices. The micro-sequence table is also read into an internal form but at this point it has not been optimized and there are some simple transformations which may be performed upon it.

The first optimization is a macro-type of substitution of subroutine calls by the instructions of the subroutine. If the subroutine is called only once throughout the micro-sequence table, then there is no reason that a call is required. The subroutine code may be placed in-line. This macro substitution optimizes in three ways, first each subroutine call is expanded into either two or four micro-ops (depending upon the type of controller used). These extra micro-ops will not be required if there is no call. Along with every call there must also be a return which requires either one or three micro-ops (again depending upon the type of controller). If all of the subroutine calls can be removed from the micro-sequence program in this manner then a much simpler controller may be used and further cost savings will be realized. There are also cases when it would be advantageous to do macro substitution even when a subroutine is called many times throughout the microprogram. If the subroutine is very short then the overhead of a minimum of three micro-ops for each call would be greater than the cost of the routine itself. Remember also that the extra hardware required to allow for micro-subroutines, when eliminated, will reduce the cost of the micro-machine. There is a hidden advantage to this type of transformation since with the subroutine instructions inserted in-line there will be potentially more parallel operations and the machine may operate faster. This is due to the fact that a call or return delimits a section of straight line micro-code and potential parallelism is only allowed between micro-operations of the same straight line micro-sequence.

A second modification which is performed on the micro-sequence table at this point is the removal of all the diverge and merge information which was included due to the way the ISP description was written. The designer determined at the ISP level that some operations were independent and that they could be done in parallel. This information would have been helpful if the allocator had been sure that the designer knew how the design would be implemented, but this is impossible. The control allocator will have to evaluate the micro-sequence program for potential parallelisms in a later step, so if the designer has guessed right then the correct information will be recovered.

The micro-sequence has a control operation called PEND which indicates the end of a particular routine. If the PEND for the main routine is executed then the machine should halt. The control allocator converts the main routine's PEND into a operation which will stop the machine. PENDs

which do not delimit the main routine, are converted by the control allocator into a return from a micro-subroutine. There is another control opcode called BAILOUT which is an instruction to cause the control flow to leave the named routine. Currently the control allocator only allows static bailouts, which are converted to branches to the end of active routine. ISPS allows for dynamic RESTARTS and LEAVES which require that the control flow leave one of the calling routines, but not necessarily at the same calling level each time it is executed. This type of control construct would require extra hardware in the controller in order to label all of the routines and it was deemed too expensive to include in any of the controller designs.

After all of these transformations have been performed upon the micro-sequence table there are likely to be cases where a JOIN instruction indicates a branch to the next instruction. A micro-step generated for this type of instruction is useless and if it is converted into a micro-op it will increase the cost and reduce the speed of the controller. To avoid this, the last step of the micro-sequence optimization is to remove such operations and clean up other instructions which would generate no-op types of micro-ops.

At this point there is enough information in the micro-sequence table to select an efficient micro-controller. Currently there is only one class of controller available, it is a microprogrammed controller with a two phase clock system. The term *two phase* refers to the types of signals which emanate from the controller and not to a system which will allow for two sequential operations to be performed from a single micro-word fetch. There are three types of micro-controllers and the proper one is selected by determining* the maximum number of subroutines which may be active at one time. A diagram of these controllers is shown in Figure 5-1. All unconditional branch addresses are stored as constants on an input to the multiplexer which is able to load the microprogram counter. The conditional branch addresses are looked up in a Rom (mcarom) when necessary. The maximum nesting level of the subroutines determines how large the micro-machine stack must be. At this point the designer must select the option of having a micro-fetch/execute overlap cycle and accept the additional costs for the extra hardware required to perform this type of operation.

5.1 Conversion of Data Operations Into Micro-ops

There are many types of data operations possible such as: binary, unary, operator, non-operator, array access, device functions and some combinations of these. These operations range in complexity from move to multiply. It is the problem of the control allocator to convert this wide range of operations into device primitives which later will be used to specify how to control the devices. This conversion is dependent upon certain aspects of the micro-sequence step itself, such as whether there has been an operator specified or if the operation has been left for one of the registers to

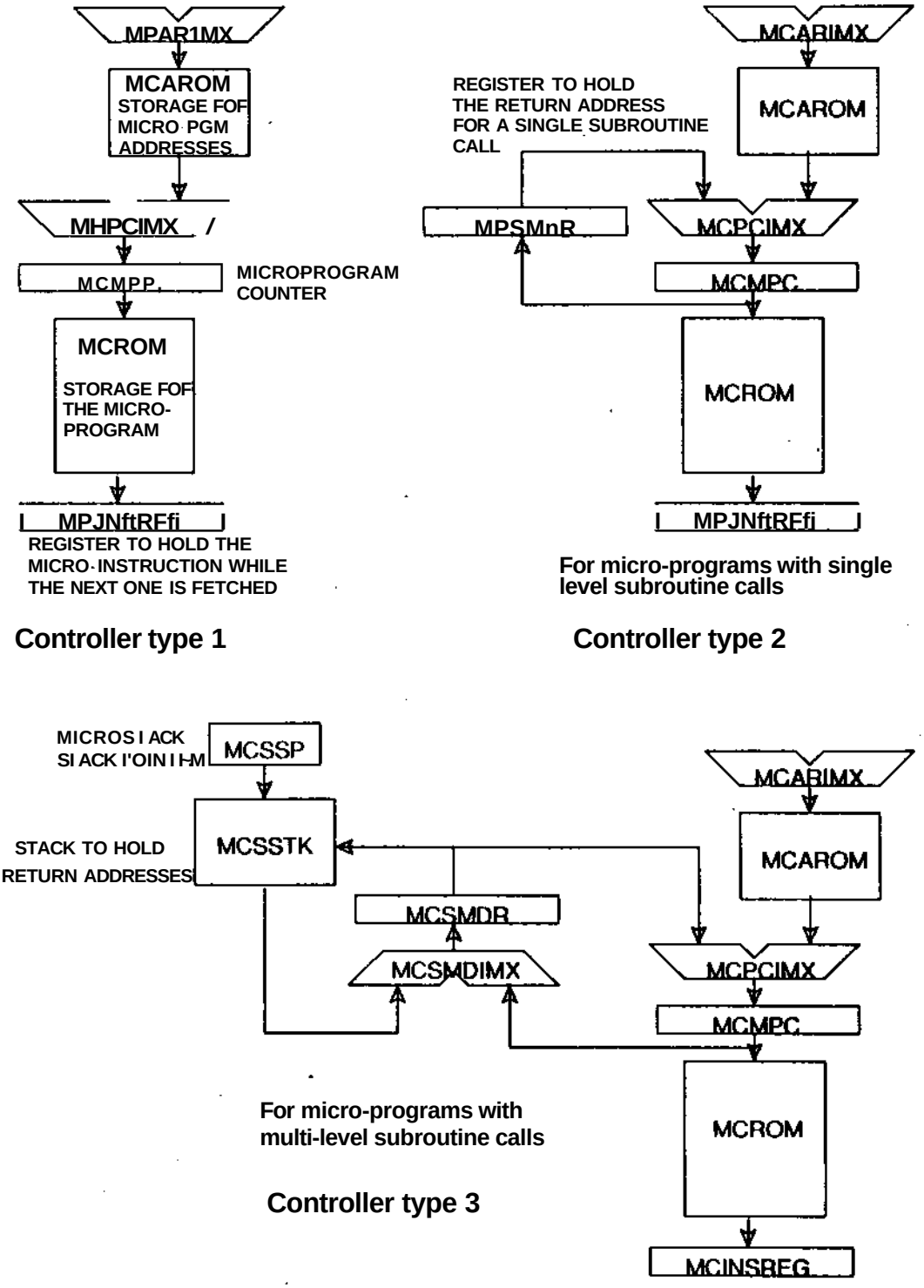


Figure 5-1: The micro-controller

perform. If there is an operator then the control allocator assigns the operation to it and the destination register is assigned the LOAD function. If the destination is not a register but instead a memory array then the operation WRITE is used. If there is no operator defined then the operation must be done in one of the source registers or in the destination register. This usually depends upon the particular opcode.

In most cases each of the source registers is assigned the operation of READ so that if there are output enable lines on that particular device they will be enabled by the controller. Since most devices are able to perform only very basic functions there is a table in the control allocator which maps the register transfer operations into basic functions. For example the add2c operation maps into a simple ADD operation. This assumes that the module binder has found a device which will do a two's complement add and has used it or it has determined that a simple adder will work (with perhaps some modification of the data paths to assure two's complement operation). Table 5-1 contains the mapping of register transfer data operations into these basic functions.

The conversion of the data micro-sequence steps into micro-ops is the first difficult step of the control allocation process. A fairly common one will make a good example:

$$\#Z43(add):\#Z:\#10_t\#3(dest)_v\#4(srcl):\#ll_v\#6(src2):\#12;$$

The opcode #243 is defined as the operation ADD. In this example the device #2 is an adder which will be used to sum the numbers stored in the devices #4 and #5. The result will be stored in the device #3. The numbers 10,1 Land 12 indicate some of the links over which the data must pass for this operation. There is however additional information not included in this instruction which must be determined from the path graph. First it must be determined if there are any unnamed devices which are used to perform this operation, such as multiplexers in the data paths. For this example let us assume that link #11 is an input to a multiplexer whose output connects to the input of the adder. The control allocator must trace through the path graph to find which input of the multiplexer is being used and store it and also remember that the multiplexer was used. All of the data path devices and their associated operations which this micro-sequence step uses are stored in a device list. Once the set of devices which this micro-sequence step requires is known then the allocator must determine how many micro-steps are required. For this the module database is consulted. In this example the allocator must find out how to cause the device #2 to ADD. If it is an ALL) then there will be some control bits to set to specific values. If #2 were a simple adder such as a 7483 then there would be no control bits and of course the micro-controller would not need to set any values for this device. There are devices and operations which may require a setup sequence, and in such a case there will be a series of steps which must be performed. Multiple control steps will cause the micro-sequence instruction to generate more that a single micro-op. Ohce the number of steps required for the micro-sequence step is known and the steps have been fixec! relative to each other then the

Register Transfer Operation			Device primitive		
test	=>	TEST	subsm	=>	SUB
eql	=>	EQL	sublc	=>	SUB
neq	=>	NEQ	mult2c	=>	MULT
lss	=>	LSS	multic	=>	MULT
leq	=>	LEO	multsm	=>	MULT
geq	=>	GEQ	div2c	=>	DIV
gtr	=>	GTR	divlc	=>	DIV
move	=>	MOVE	divsm	=>	DIV
clear	=>	CLEAR	mod2c	=>	MOD
noop	=>	NOP	modlc	=>	MOD
read	=>	READ	modsm	=>	MOD
write	=>	WRITE	mult	=>	MULT
lshftd	=>	SHIFTL	div	=>	DIV
rshftd	=>	SHIFTR	mod	=>	MOD
lrot	=>	ROTL	amove	=>	ALOAD
rrot	=>	ROTR	amove2c	=>	ALOAD
not	=>	NOT	amovelc	=>	ALOAD
incr	=>	INC	amovesm	=>	ALOAD
deer	=>	DEC	test2c	=>	TEST
and	=>	AND	eql2c	=>	EQL
or	=>	OR	neq2c	=>	NEQ
nand	=>	NAND	Iss2c	=>	LSS
nor	=>	NOR	Ieq2c	=>	LEQ
xor	=>	XOR	geq2c	=>	GEQ
eqv	=>	EQV	gtr2c	=>	GTR
add	=>	ADD	testlc	=>	TEST
sub	=>	SUB	eql lc	=>	EQL
lshftl	=>	SHIFTL	neqlc	=>	NEQ
rshftl	=>	SHIFTR	lsslc	=>	LSS
lshftO	=>	SHIFTL	leqlc	=>	LEQ
rshftO	=>	SHIFTR	geqlc	=>	GEQ
cone	=>	LOAD	gtrlc	=>	GTR
neg2c	=>	NEG	testsm	=>	TEST
neglc	=>	NEG	eqlsm	=>	EQL
negsm	=>	NEG	neqsm	=>	NEQ
add2c	=>	ADD	lsssm	=>	LSS
addle	=>	ADD	leqsm	=>	LEQ
addsm	=>	ADD	geqsm	=>	GEQ
sub2c	=>	SUB	gtrsm	=>	GTR

Table 5-1: The mapping from register transfer instructions to device primitives

micro-op(s) may be generated. The steps are fixed in a relative position because if one of the devices requires a single step and another requires three they must be evoked at the same time.

For each, micro-op a device list is generated which contains only those devices which the micro-op needs. Each element of this list contains n flaq indicating whether the device is required as a source

or destination which will be important during the potential parallelism analysis step.

There is an important feature of the micro-sequence table which might not be apparent to the reader. From the above description one might think that only one operation may be performed during each micro-op. This is not true. A micro-op is only generated when a micro-sequence step indicates that there should be an evoke signal. This is designated by the presence or a destination in the micro-sequence step or the execution of a control operation. If there is nothing to evoke then the micro-sequence step is called a chained micro-sequence step and all of the control information which it generates is saved and added to the immediately following step (which may itself be a chained micro-sequence step). The number of steps in a chain is unlimited so that any combination of operations may be performed in a single micro-operation.

5.2 The Conversion of Control Operations Into Micro-ops

The conversion of the control operations into micro-ops is not as simple a process as it was for data operations. The allocator must convert each control operation into a data-transfer operation which operates upon part of the micro-controller itself. The control operation JOIN is the simplest; all that is necessary is that the new micro-address be jammed into the microprogram counter. If the control operation is a conditional branch then the new address must be looked up in a table (rom) and then placed in the microprogram counter. For calls to subroutines the old microprogram counter must be stored (either in a register or on a stack) and the new one placed in the counter, while for returns the old microprogram counter must be restored and the stack pointer changed. The action of control operations is highly dependent upon the type of controller which has been selected and for this reason the device primitives are built into the control allocator program.

The CALL micro-sequence instruction is a good example of the necessity of building these device primitives into the control allocator. If a type 2 controller (Figure 5-1) is selected then the CALL micro-sequence instruction is converted into two micro-ops:

```
iftcro-op 1  =  Load MCSMDR                    ISave the old micro program address

Ificro-op2  =  Select proper input of MCPCIMX
               Load MCMPC                      ILoad the new microprogram address
```

If the controller selected was of the type 3 form then the call micro-sequence step is converted into four micro-ops:

```
Micro-op 1  =  Increment MCSSP                    Nncrement the micro stack pointer
Micro-op 2  =  Select input # 1 of MCSMDIMX
               Load MCMDR                         !Save the old microprogram address
Micro-op 3  =  Write MCSSTK                       IStore old value on the stack
Micro-op 4  =  Select the proper input of MCPCIMX
               Load MCMPC                         ILoad the new microprogram address
```

In this example micro-op 1 for the type 2 controller and micro-op 2 for the type 3 controller perform the same function but have different device primitives because in the type 3 controller there is an extra multiplexer which must be controlled.

5.3 Micro-cycle Time Evaluation

Once a complete device list for a micro-op is available then the control allocator may determine the time required to perform each micro-op. This timing measurement is necessary in order to determine the optimal micro-cycle time. The minimum allowable micro-cycle time is defined as the maximum of the following:

- The maximum time it takes to read a word out of the microprogram store when using the microprogram counter as an address.
- The maximum time it takes to determine the next micro-address by accessing the micro-address rom.
- The maximum time required to retrieve the next micro-address when a micro-subroutine call or return is being executed.
- The minimum time required to perform any of the data operations in the data path section of the machine.

The selection of the micro-cycle time has been left to the user but the control allocator measures all of the operation times and reports the maximum and minimum times for the user to use in his selection. In order to evaluate the time required for each micro-operation the control allocator must find the slowest path over which data must pass from any source device to any destination device. To do this the allocator evaluates the time required for data to propagate from each devices output to the output of the device which follows it. The control allocator then finds the slowest path from each of the source devices to a destination device and takes this as the operation time for this micro-operation. The control allocator allow for the user to specify the desired micro-cycle time. When this has been done the micro-operation program is reviewed and any micro-op which requires more time than the specified cycle time will be divided into a series of micro-ops. Each of these will fit into the specified micro-cycle time slot and each performs part of the original operation.

5.4 Control Graph Generation

At this point, the design description is a sequential list of micro-operations. It is highly probable that this program contains at least two sequential micro-operations which have disjoint device sets, and in most cases these two micro-ops may be executed in their original order or in a reversed order without affecting the results of the program. Since these operations are hardware-independent they may even be done at the same time with equivalent results. Two micro-operations are defined as potentially parallel if the execution of both in parallel would cause the same results as when they are executed in the original order. Potential parallelism is not limited to the combination of just pairs of operations. If three micro-ops are all device independent and their execution ordering is non-consequential then all three are defined as potentially parallel. The combination of serial micro-ops into parallel operations will reduce both the length of the microprogram and its execution time, and the control allocator will attempt to exploit such potential parallelism because of these advantages.

In the previous paragraph, two rules were used to determine if micro-ops were potentially parallel. They were, total device independence and execution order insensitivity. These are valid rules but also excessively restrictive. Another set of rules is presented by Dasgupta [Dasg 76]. His rules indicates that two micro-operations, MO_a and MO_b are potentially parallel if the following is true:

$$(SC_a \cap SK_b = \phi) \wedge (SC_b \cap SK_a = \phi) \wedge (SK_a \cap SK_b = \phi) \wedge (U_a \cap U_b = \phi)$$

Where:

SC_i = The set of source devices for micro-operation *i*
SK_i = The set of sink devices (destinations) for micro-operation *i*
U_i = The set of paths (links and operators) used in micro-operation *i*

This rule simply states that two micro-operations may be done in parallel if one's source is not the destination of the other, and that they do not use the same links or write to the same destination. It is slightly more comprehensive than the one presented at the beginning of this section because it also considers the links and operators which the data must pass through. This rule does not, of itself, detect cases where the interchanging of the order of execution would generate an incorrect result, which may indicate that parallel operation would also be incorrect. To avoid such cases Dasgupta only applies the rule to micro-ops which are in the same Straight Line Micro-code Segment. A SLMS is defined as a section of the micro-operation program which has no branches in it. An equivalent definition is: a section of the micro-op program which only contains data micro-ops (ie. no control ops). If two congruent micro-ops of a SLMS satisfy Dasgupta's rule then they are potentially parallel.

Dasgupta's rule is correct but still too restrictive to be used by the control allocator. There are

usually cases where two micro-ops use the same sources but different destinations and it is possible to do both in parallel. His rule might not indicate that these were potentially parallel because of the $(U_a \cap U_b) = \langle\rangle$ restriction. The optimal rule for potential parallelism detection indicates all micro-ops which may be done in parallel with the available hardware and disallows combinations which would produce incorrect results. This rule is dependent upon the controller which is used and a new rule must be defined whenever a new controller style is implemented.

The rule used for this project is defined as follows:

For two micro-operations of the same SLMS where MO_i precedes MO_f the micro-operations are potentially parallel if the following is true.

$$(SC_i \cap SK_f = \langle\rangle) \wedge (SK_i \cap SK_f = \langle f \rangle)$$

and

If $(SC_i \cap SK_f \neq \langle t \rangle)$ then SAME, «- MO_f is added to MO_i's SAME list

and

If $(SU_i \cap SU_f = C * \langle f \rangle)$ then the following must also be true, for all k: $C_{kfj} = C_{kfi}$

Where:

SU_{iT}	=	$SC_i \cup U_{ir}$ This is the set of all devices used in the micro-op except for the destination devices.
$SAME_n$	=	A pointer in MO_i to the micro-operation(s) which it must never occur before.
C	=	The intersection of the source sets of the two micro-ops.
$C_{n_{fn}}$	=	The function (device primitive) specified for the device C_n by MO_{fT}.

In simple terms this rule states that two micro-ops are potentially parallel if the preceding micro-op's destination device(s) are different from the other's sources, and their destination devices are different, and if they use any devices in common (SU_i) then these devices must be controlled in the same manner for both operations.

The availability of the $SAME_i$ pointer allows for cases where the interchanging of the micro-ops would affect the results of the program but their execution in parallel will yield correct results. The SAME pointer does not necessarily point to a single micro-op, it may also point to a list of micro-ops which the micro-op must never occur before.

The use of this potential parallelism rule may be demonstrated with a couple of examples:

Example 1

```
MO-1    A«-B
MO-2    C*-D
MO-3    E<-F
MO-4    D*-A
```

In this example MO-1 and MO-2 are device independent so they are potentially parallel. The same is true for the combinations of MO-2 and MO-3, MO-1 and MO-3, and MO-3 and MO-4. Note that it is not true that when MO-1 and MO-3, and MO-3 and MO-4 are potentially parallel that MO-1 and MO-4 are also potentially parallel. In this case the result of MO-1 is required by MO-4 and so they must be sequential. In this example it is also true that MO-2 and MO-4 are potentially parallel but for this case $SAME_4$ is set to point to MO-2 indicating that MO-4 must not occur before MO-2.

Example 2

```
MO-1    D*-25
MO-2    B*-L
MO-3    A<-B+C
MO-4    L*-3
MO-5    C«-D
```

In this example the following combinations are potentially parallel: MO-1 and MO-2, MO-1 and MO-3, MO-1 and MO-4, MO-3 and MO-4, MO-4 and MO-5, MO-2 and MO-5. There are also of potentially parallel pairs of MO-2 and MO-4 in which $SAME_4$ points to MO-2, and MO-3 and MO-5 where $SAME_6$ points to MO-3. The combination of MO-2 and MO-5 is of little interest since it will never be allowed to be parallel because $SAME_5$ points to MO-3 which must follow MO-2.

5.4.1 Application of the potential parallelism **rule**

The procedure which is used by the control allocator is the fastest possible method which will test for all potentially parallel operations. An approach which just compares each micro-operation with all the others in the same SLMS is inefficient and rather expensive since the number of comparisons grows polynomially as a function of the number of micro-ops in the SLMS.

The method used by the control allocator takes advantage of the fact that the micro-operation program is ordered and that some comparisons may be ignored when certain conditions are found to be true. These conditions may be seen in the following example.

MO-1 A«-B+C
MO-2 D*-A+K
MO-3 L«-A
MO-4 W<-D

Consider MO-4; comparing it with MO-3 they are found to be potentially parallel. Next, comparing it with MO-2 it is found to not be potentially parallel. There is then no need to compare MO-4 with MO-1 since MO-4 must follow MO-2 and the relationship between MO-2 and MO-1 will determine how MO-4 and MO-1 will be related. The control allocator avoids the unnecessary comparisons by remembering cases where this data dependency precludes any potential parallelism.

This procedure used by the control allocator is one of finding the range of potential parallelism for each micro-operation. To do this first a base micro-operation is selected and it is compared with its immediate predecessor. If these two are potentially parallel then the original micro-op and its predecessor's predecessor are compared. This process continues to backtrack through the micro-op program until a limit is found. A limit is defined as any one of the following: the beginning of the micro-operation program, a micro-operation which is not potentially parallel with the base micro-op, or the beginning of the SLMS. Once a limit is found an entry is made in a table which stores lists of micro-ops which may be started immediately after each upper limit micro-op is completed. During the search for the upper limit the SAME pointer may have had entries added to it. This does not constitute an upper limit but the information about these listed micro-operations will affect some of the potential parallelism in a later step.

The next step is to find the lower limit of each micro-operation. To do this a base micro-op is again selected and successive following micro-ops are compared with the base until a limit is found. The lower limit is defined as one of the following: the micro-op which is not potentially parallel with the base micro-op, the end of the SLMS (the micro-op which follows the first control operation[2]), or the end of the micro-operation program. The base micro-op is entered in a second table which contains lists of micro-ops which must be completed before the lower limit micro-operation may be started.

This procedure is repeated for each micro-op in the program and when completed there are two tables, one called the UPPER and a second called the LOWER, which contain all of the potential parallelism information. For the control allocation project a graph structure was selected to represent this information and the next step of the control allocation process is the generation of the Control graph from these tables.

---

[2] A slight modification of the definition of SLMS has been made for use by the control allocator The change is that the last micro-operation of a SLMS may be a control operation. This allows for a control micro-operation to be performed in parallel with a regular data o|>eration. This type of overlap is almost always taken advantage of in microprogrammed machines because of the extra time which is required to generate the new microprogram address.

5.4.2 The Control Graph **Model**

The control graph is a representaion of the potential parallelism information which was determined in the preceding step. It only represents the potential parallelism information and does not include any control flow information (such as control ops) which one might normally associate with a control graph. The control operations such as the IF statement were converted into data transfer operations during the generation of the micro-operation program and their effect has not been lost but they do not exist as control ops in the control graph.

There are three type of nodes in the control graph, Forks, Joins and Operations. The Operation node is the most common and is defined as:

Operation Node = <Forward,Backward,Sequence number,Devicelist,Cgsame>

The fields Backward and Forward are pointers to other control graph nodes which must precede and follow (respectively) this node in order of execution. With these fields the nodes of the control graph are interconnected. The Sequence number is both a sequence number and a cross reference to the micro-operation which this control graph node was generated from. Since there is a one to one relatlonship between each control graph node and each micro-operation this number is unique for each control graph node. The Devicelist is a list of the devices which this control graph node must control. It is a subset of the device list which is contained in the micro-op and does not contain those devices which have no control lines. Cgsame performs a similar function as that done by SAME in the micro-ops, it points to the control graph node(s) which the current control graph node must never occur before.

The fork nodes of the control graph indicate the start of two or more potentially parallel paths in the control graph. It is defined as:

Fork Node = <Backward,List of followers>

The field Backward points back to the preceding control graph node. The list of followers is a list of the control graph nodes which must follow this one.

The join nodes indicates when two or more potentially parallel paths of the control graph must be completed before another node may be started. The form of the join node is:

Join Node = <Forward,List of Predecessors>

The Forward field points to the control graph node which follows this node and the List of Predecessors contains a list of the control graph nodes which must be done before this one is started.

There is nothing in the control graph structure which requires that the fork and join nodes be

paired. A control graph may contain a single fork which has ten potentially parallel legs, and four joins which "collect" these legs when they are no longer potentially parallel. The forks only indicate when potential parallelism starts, and the joins when it ends.

The generation of the control graph is a simple process once the UPPER and LOWER tables are complete. The first step is to provide a control graph node for each micro-op in the program. The forward and backward pointers are left empty at this time. The UPPER table is then processed. For each entry in this table which has more than a single follower a fork is inserted in the control graph which will branch to each of the followers. The pointers in the fork node point to all the correct places so that the upper limit control graph node (and micro-op) are completed before the following nodes are started. In cases where there is only a single follower then only the forward and backward pointers of the affected control graph nodes are set. After all of the UPPER table has been processed the LOWER table is used to place the joins in the control graph. For each entry in the table which has more than one predecessor a join from the listed nodes is inserted in the control graph. When this table has been finished all of the Forward and Backward fields will have been filled and the control graph will be complete. Figure 5-2 shows the process of generating a control graph for the microprogram used in Example 1 above.

The information which is contained in a control graph can be very confusing when it is not represented in the graphical form. To demonstrate this and to further understand the power of the control graph structure consider the control graph shown in Figure 5-3. If one does not consider the constraints caused by the Cgsame pointers then there are 38 potential parallel combinations of two micro-ops each in this example. A few of these are the combinations of 1 and 2_T 2 and 10, 3 and 9, 5 and6, and 5 and 7. Combinations which are not allowed are ones such as 1 and 10, 2 and 9, or 4 and 7. When one considers the limits set by the Cgsame pointers then there are only 30 potential parallel combinations of two nodes. This is caused by the Cgsame of node 11 which indicates that it must never occur before node 9 and thus it is not really potentially parallel with nodes such as 5 and 7. Note that node 11 is still potentially parallel with node 3 since node 9 and node 3 are potentially parallel and the nodes 11 and 9 are also potentially parallel.

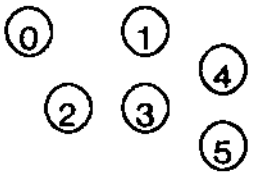
5.5 Micro-instruction Definition and Micro-word Formatting

This step of the control allocation process was not developed by the author of this report but since it is an integral part of the process it will be outlined here. For a more detailed description of this step and the problems associated with it see [Nagl 78].

The control graph contains a representaion of all of the potentially parallel combinations of micro-ops which may occur. Since it only represents the potentially parallel combinations a decision

Original micro-op program

MO-1	A<- B
MO-2	C«- D
MO-3	E<- F
MO-4	D«- A



Step 1

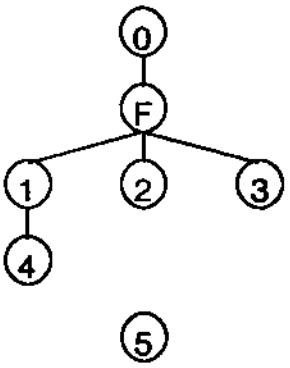
Generation of the CG nodes without the forward and backward fields filled in.

Upper table

Micro-op node numbers	Micro-op(s) which must follow
MOO	1,2.3
MO-1	4
MO-2	
MO-3	

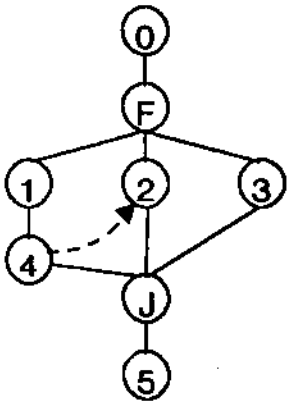
Lower table

Micro-op node numbers	Micro-op(s) which must precede
MO-1	
MO-2	
MO-3	
MO-4	1
MO-5	2,3,4



After processing the Upper table

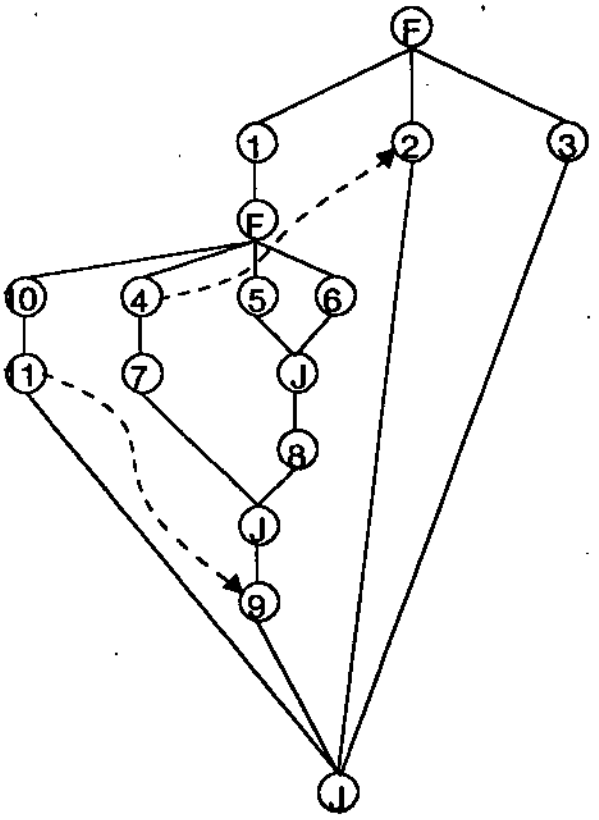
Step 2



After processing the Lower table

Step 3

Figure 5-2: Steps in Control Graph generation



The solid lines indicate the same information which is contained in the forward and backward fields. The dashed lines indicate the the nodes which Cgsame points to. The numbered nodes are the ordinary control graph nodes. The forks and joins are represented by F and J respectively.

Figu re 5-3: An example control graph

must be made about which of the micro-ops will actually be performed in parallel. This decision is **not** trivial because in most cases a micro-op may be potentially parallel with more than one other micro-op and one combination may be better than the other. One method which may be used is to arbitrarily combine micro-ops into micro-instructions. Operating in this manner it is possible to take advantage of all of the potential parallelism of the micro-op program and thus generate the shortest possible microprogram but the width of each micro-instruction may be wider than the width necessary if one had been careful about combining the micro-ops.

In microprogrammed machines the width of the micro-instruction is a function of how the instruction is formatted. In machines which have a horizontal word format each bit of the micro-instruction contains the signal for a single control point of the digital system and the micro-instruction has enough bits so that every control point may be controlled in a single word. Obviously such words tend to be very wide and in such cases the micro-storage memory for a system is a substantial part of the total controller cost. In machines which have a vertical word format the micro-instruction contains fields which are decoded by the controller to generate the proper control signals. A totally vertical machine has the control signals encoded so that only a single micro-op may be performed by each micro-instruction. The advantage of encoding the micro-instruction is that the width of the instruction is reduced which tends to reduce the cost of the microprogram memory. A disadvantage is that only a single micro-operation may be performed during a micro-instruction and thus any potential parallelism is not possible.

In most designs it is highly unlikely that there will ever be a need for a micro-instruction which will be able to evoke every device at once (for which the horizontal format is required). It is also unlikely that there will ever be a micro-operation program with no potential parallelism, for which a totally vertical format would be the correct choice. Since neither extreme is expected the obvious solution is to use a hybrid format which combines the advantages of both. This new format, which the control allocator uses, allows for all the parallel micro-operations necessary but it also overlays or encodes some bits to reduce the width of the control word. To use this new format care must be taken when defining the micro-instructions so both the instruction definition and the formatting processes must work together to reduce the cost of the controller while trying to generate the fastest possible microprogram. The reason that this factor may be exploited by the control allocator is that the controller has not been defined and the allocator may decide what formats the micro-instructions will have. In systems which compile micro-instruction for predefined controllers the format of the micro-instructions are already specified and the combination of micro-operations is limited to the ones which will fit into the available formats.

The process of determining which micro-ops will be done in parallel can be thought of as a kind of mental exercise with an imaginary control graph. Consider a control graph such as the one in Figure 5-3. The control graph nodes in this imaginary control graph are connected to each other by elastic links. The nodes of the graph are allowed to move only up or down on the page and when moved the elasticity of the links maintains their connections. The dotted arrows of the control graph (the cgsame pointers) are special elastic links which must never have a negative slope. A negative slope is defined as when the arrowhead is lower on the page than the tail. The time axis of this system starts at the top of the page and the positive direction is towards the bottom of the page. Nodes which are above others are performed before them when the control graph is converted into actual instructions. The

objective of this exercise is to move the control graph nodes relative to each other to see how they interact and eventually to determine the best set of micro-instructions. A micro-instruction is defined as any number of Cgnodes (not including the fork and join nodes) which lie on a horizontal line. For the control graph of Figure 5-3 it is possible to configure the graph so that up to six micro-operation are on the same horizontal line and thus in the same micro instruction. Figures 5-4 and 5-5 show two other possible configurations of this same graph.

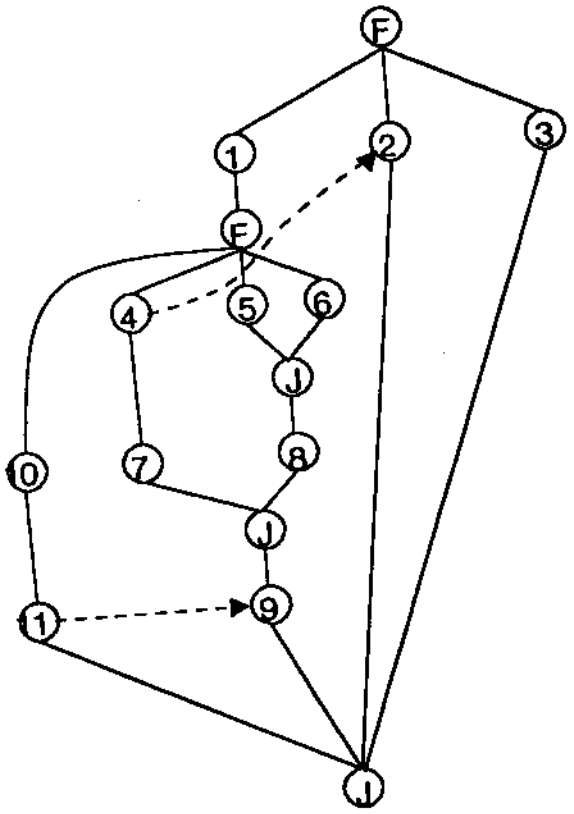
The method which the control allocator uses to select micro-ops for the micro-instructions is based upon attraction weights. These weights are similar to the probability that two potentially parallel micro-ops will be combined. The attraction weights are calculated for every combination of micro-ops throughout the micro-op program and the pair with the highest weight are placed in the same micro-instruction first. It is then necessary to then recalculate the potential parallelisms and attraction weights since the action of combining two may restrict the range of other micro-ops in the program. When the new attraction weights are know the pair with the highest value are combined and the process is repeated until there are no more potential parallel pairs or the size of the micro-word is too wide. If the width is the limiting factor then the routine creates a new micro-instruction format and continues the process. With this procedure the micro-instructions will contain collections of micro-ops which occur in parallel the most often throughout the complete micro-op program.

5.6 Control Signal Conditioning and Micro-word Representation

When all of the micro-instructions have been defined only a few minor details remain which the control allocator must address before it is done. The first is that the controller must be defined in such a way that the two phase control signals which were assumed to be available in an earlier step will be generated. To do this the control allocator must specify whether an AND or NAND gate should be used to condition the phase 2 clock pulse for each of the evoke control points in the design. The value of the NONEVOKE[3] field for the particular control point, stored in the Module Database, is the determining factor. For each control point which has a NONEVOKE value of H(high) a NAND gate is specified and for each with a L(low) value the AND gate is specified.

If during the micro-word packing step more than one micro-instruction format was specified then the control signal conditioning will also include the bit steering decoder which is placed between the micro-instruction register and the control points of the design. This device just directs the evoke control signals stored in the micro instruction to the proper devices of the data path and is itself

[3] Refer to section 4.2 for a description of what this field represents



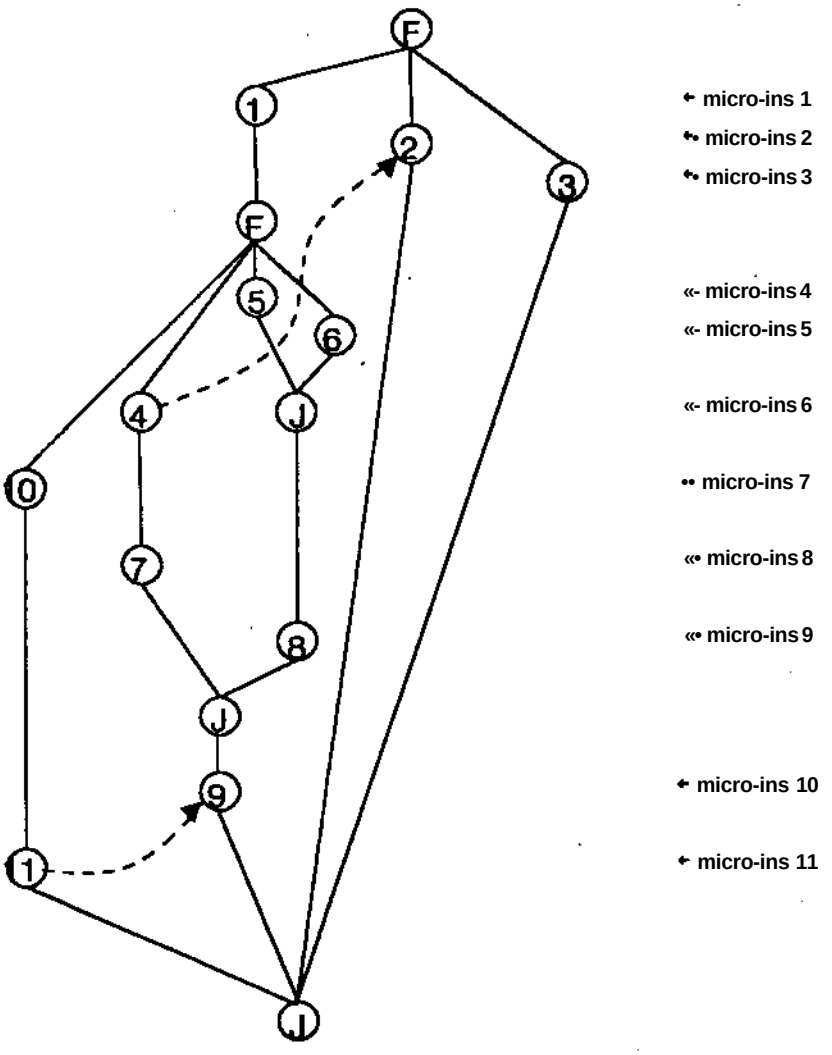
«- micro-ins 1

«- micro-ins 2

«-micro-ins 3

«- micro-ins 4

Figure 5-4; A modified control graph



The longest possible program, one micro-op per micro-instruction

Figure 5-5: Another modified control graph

controlled by a field of the micro-instruction.

One additional detail which must be considered is how to represent the micro-instruction in a form which may be used by the builder of the digital system. The micro-instructions are internally represented as collections of micro-ops which are just lists of device primitives. These primitives are converted by the control allocator into the actual ones, and zeros which should be placed the microprogram rom. To convert the device primitives into binary values the control allocator compares the BITVAL for a particular control line of the CTLESEQ for the current device primitive to the NONEVOKE value for the device. If these values are not the same then the value to be stored in the microprogram rom is a one, if they are the same then the value is zero. This comparison is performed for each control line of each micro-instruction and a table of these ones and zeros is printed in the same file as the data path graph and the micro-sequence table. A table similar to this is also printed which contains the programming for the micro-address rom. For this memory the values are just the addresses in the microprogram which will be branched to during select instructions.

In the binary form the microprogram is difficult for a human designer to understand, so the control allocator also provides an optional file which contains an english version of each micro-instruction. Since each instruction was generated from a micro-operation which was itself generated from a micro-sequence step they are all related and the control allocator prints the micro-sequence step for each micro-instruction so that the user may relate it back to the original program.

6. Results

During the process of building the control allocator a few features of the control allocation process became apparent. The first was that it is absolutely necessary that the data path graph be completely bound before the micro-op program generation step can begin. The path graph must also contain the hardware which makes up the controller. Unfortunately, details such as the size of the microprogram are unknown at this early stage of the design so the control allocator must estimate the requirements in order to bind the devices to physical hardware.

The second result was that the structure of the control graph allows for every potentially parallel combination of micro-operations to be described. The control allocator itself always finds any potential parallelism in a SLMS and so the control allocator using the control graph does not introduce any restriction on the speed or cost of the digital design. If it had restricted the design then one could not expect optimal designs from it.

In an effort to compare the controllers designed by the control allocator and the ones designed by humans, a path graph which nearly matches the data paths of the PDP11/40[4] was generated by hand [5]. This path graph was then processed by the control allocator and the size of the generated microprogram was compared with the one in the human-designed PDP-11/40. A problem in the representation of digital systems was found which indicated a need for extensions in the data path graph "language". It is presented here because the lack of specific features constrained the possible designs which the control allocator was able to develop.

Thus, a third result is that there is a genuine need to be able to specify control lines in a data path graph. This is not necessary when the path graph is being generated automatically but when one wants to hand-code a path graph for an existing system, there may be some aspects of it which cannot be described. In the PDP-11/40, for example, the ALU is in some cases controlled from the micro-machine and in other cases its function is specified by decoding the instruction register. In the machine-generated data path graph the control signals are the responsibility of the controller, so one method to implement such an ALU control problem would be to leave the problem of connecting the instruction register to the control lines of the ALU up to the control allocator. This would seem to be a reasonable solution but there is no way to specify such a connection in the micro-sequence program. Even if there was such an instruction, the data path allocator would be unable to completely specify it (the data path allocator writes the micro-sequence program) since it does not even know that an ALU

[4] PDP is a registered trademark of Digital Equipment Corporation

[5] A diagiam of this Data Path and a listing of the Micro Sequence program are included in Appendix 1

would be used to perform the desired function and that the ALL) would require control signals.

The comparison of the automatically generated controller and the PDP-11/40 controller is incomplete but the preliminary results indicate that the control allocator is performing well. In this comparison an effort has been made to closely match the data paths of the human-designed PDP-11/40 but to allow the control allocator to generate its own controller design. The optimization routines which were written by Nagle were used to generate the fastest possible microprogram **for the** micro-sequence specified [Nagl 78].

In one experiment 36 PDP-11/40 micro-words were implemented in a micro-sequence program. These words consisted of the macro-instruction fetch and the source and destination processing sections of the microprogram. The program was added to the data path graph for the 11/40 and run through the control allocator. The resulting automatically generated microprogram was 38 words long and 62 bits wide. To relate these numbers to the human design one must note that it has a micro-word width of 56 bits but that not all of these should be included in the comparison. The PDP-11/40 micro-word fields called CLK, CB, CBA_t and CD (which require a total of six bits) should not be included because they are control lines for clock signals which were not implemented in **the** data path description. With these bits removed the PDP-11/40 micro-word is 50 bits wide which is still a fairly close match to the 62 which the control allocator designed. Thus the size of the automatically generated controller in total bits required was about 31% larger than the human design. This comparison looks even better if one realizes that the PDP-11/40 is able to perform more than one micro-op in a single micro-instruction and so the number of operations which were packed into 38 words of the control allocator's microprogram was potentially 72 and in actuality 44 micro-operations [Nagl 80].

A user of the control allocator should be primarily interested in the resulting controller designs **but** consideration should also be given to the speed with which each design is generated. The **control** allocator was run for a few different digital designs and the execution speed of specific sections of **the** design process were measured. The execution times were recorded for the processes of micro-operation program and control graph generation. Figure 6-1 is a graph of the results of these runs. The four designs which were used in this test were the AM2909, MARK1, AM2910 and the PDP-11/40. The first three designs were automatically generated and the path graph for the PDP11/40 example was hand generated.

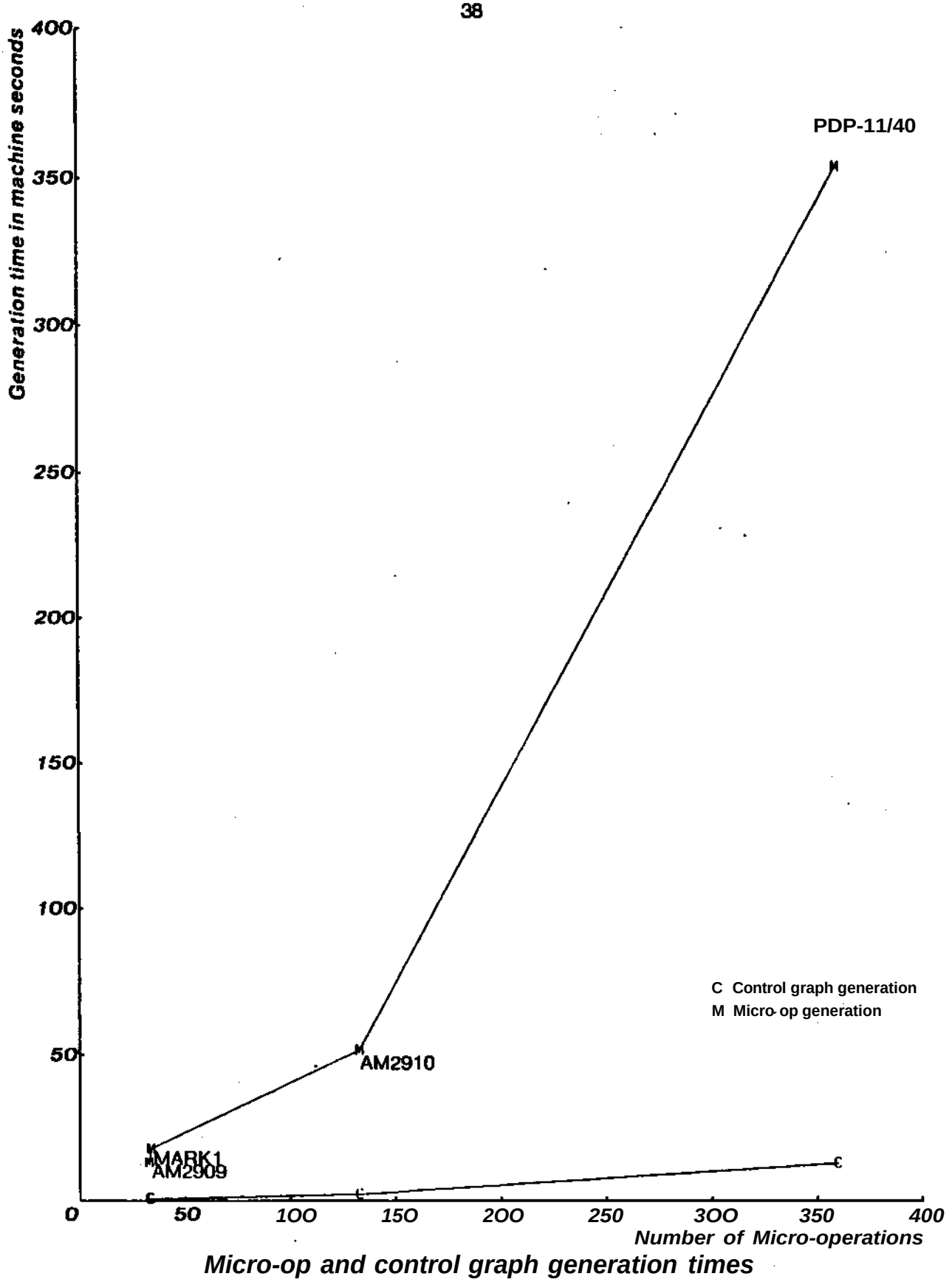


Figure 6-1: Micro-op and control graph generation times as a function of micro-ops

7. Suggested Improvements for the Automated Control Allocator

During the evolution of the control allocator some aspects of the problem have become apparent which were not investigated due to their difficulty or the lack of time to properly address them. One of the primary problems of the control allocation process which could be improved is the method by which the micro-cycle clock speed is selected. Currently the times required for each micro-op are calculated and the minimum and maximum are reported. The user then specifies a time which he feels would be the best. A better solution to this problem would be a system which provides a histogram of the micro-op execution times and the user would then select the cycle time which would cause the least amount of wasted time. Even this method may be improved upon, however. One can not assume that each micro-op will be executed with the same frequency in a digital system, since some operations such as the instruction fetches will occur more often than instructions such as halts. A system which measures the relative frequencies of each micro-operation and combines these values with the operation times should be able to determine a micro-cycle clock speed which will result in an optimal controller design.

Since the control allocator is only a tool and repeated processing of slightly modified digital designs is probable it would be helpful if the control allocator could find sections of the micro-sequence program which cause bottlenecks in the control graph. This information could be used by the data path allocator to make changes in the data path structure so that the slower sections of the micro-sequence program may be improved. Along the same lines the control allocator could identify the execution speed of macro instructions of the digital system and report these so that the slow ones could be speeded up by either modifications to the data path or instructions to some of the other control allocator routines.

A third problem is that the current control allocator is only able to generate a controller for a single process executing on a data path. In ISPS the designer is able to describe parallel independent processes and the data path allocator is able to define the hardware for such ISP descriptions. The control allocator is not able to handle this type of system and will not resolve or even recognize any hardware conflicts which may arise. To solve this problem the control allocator would have to define the additional control hardware which will do the arbitrating for it.

The current control allocator is limited by the restriction that multi-phase micro-instructions are not allowed. This type of instruction format would allow for a higher density packing of micro operation in the micro-instructions but also is much more difficult to process. An improved version of the control allocator would design multi-phase-controllers which would be able to generate multiple sequential

operations from a single micro-instruction fetch. Such a system has the advantage that the speed of the microprogram memory can be slower (and usually less expensive) without slowing down the execution speed of that program.

8. Conclusions

The ideas and results presented in this report should lead the reader to the following conclusions.

- The automated generation of microprogrammed controllers from the CMU-DA data path graph is a reality.
- The automated generation of controllers from any hardware description language is a very strong possibility since the techniques presented here may be adapted to other design automation programs.
- The control graph structure which is used to represent the potential parallelism of **the** data paths does not restrict the design in any way.
- The control allocator is able to generate microprograms which are similar in size to those which have been extensively hand optimized.

References

[Dasg 76] Dasgupta, S.
Parallelism in Microprogramming systems.
PhD thesis, Department of computer Science, University of Alberta, August, 1976.
Tech. Report TR76-7.

[DeWi 76] DeWitt, D. J.
A Machine-Independent Approach to the Production of Horizontal Microcode.
PhD thesis, University of Michigan, August, 1976.
Tech. Report 76 DT4.

[DeWi 78] DeWitt, D. J.
The Complexity of Microprogram Optimization.
1978.

[Leiv 79] Leive, G.W. and Thomas, D.E.
The CMU Design System, Module Database-Users Guide
1979.

[Mall 78] Mallett, P. W.
Methods of Compacting Microprograms.
PhD thesis, University of Southwestern Louisiana, December, 1978.

[McFa 78] McFarland, M.C.
The Value Trace: A Data Base for Automated Digital Design.
Master's thesis, Carnegie-Mellon University, December, 1978.

[Nagl 78] Nagle, A.
Automatic Design of Micro-controllers.
In *Proceedings of the 11th Annual Workshop on Microprogramming.* IEEE Computer Society, 1978.

[Nagl 80] Nagle, A., R. Cloutier,A. Parker.
Future Article on Control Allocation.
IEEE Transactions on Computers , 1980.

I. Appendix 1: The PDP-11/40 example

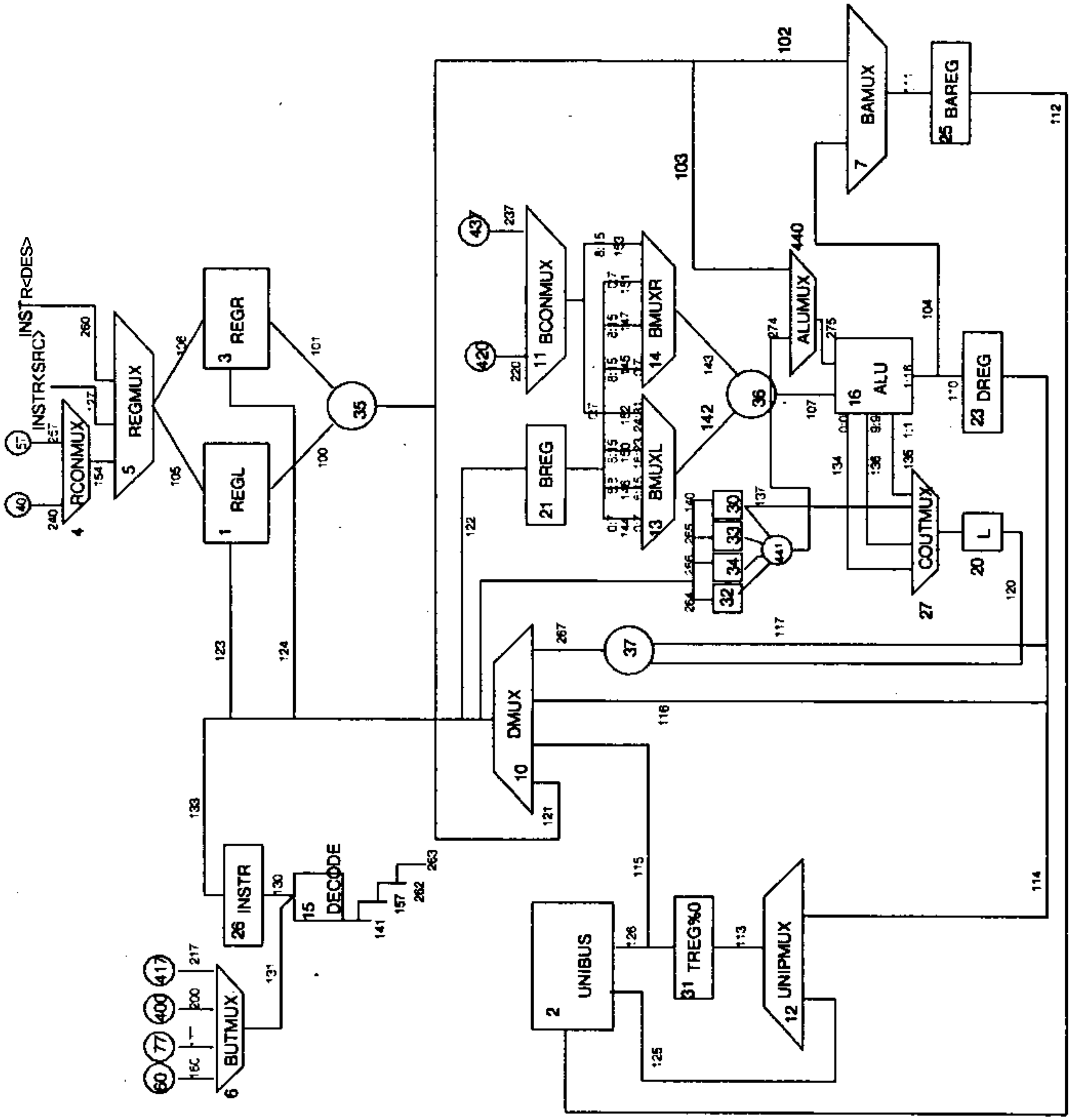


Figure 8-1: The [illegible] diagram used to [illegible] the PDP-11/40 in microprogram size experiments

This is the hand generated micro-sequence table for the PDP11/40. It represents 185 of the micro-words of the human designed machine.

```
        ****** MICRO 0PIRA110N SI QUINCE ******
-----------------------------------------------------------------------

1#1:    #352(P1HG1N).#77 7(PI)P114O). '000000000;
        #22O(RIAD):#10#3,,#47(07):#247.;
        #21O(MOVF).#25(BAIUG).#10#3(RfGI0IUGR):#IOOS#IOI,;
I#133:  #?20(RI AD).#31(TRI G70),#2(UNIHUS),#25(BAREG);
        #210(MOVF),#26( INSTR),#31(1RIG'X.O):#115, ;
        #210(MOVI ).#21(HRIG).#31(TRrGX0):#115.;
        #22 1 (Win II ),#10#3(RIGI 0IUGR).#53( 13) .#31(TREGX0);
        #220(RI AI)):#10#3..#47(07 ):#247.;
        #200(II SI):#36:#1520#153,,#422(2):#222.;
        #243(ADD):#16:#1O4.#25(HAREG).#10#3(REGL0REGR):#1OO0#1O1.#36(CONCAT):#1O7;
        #220(RI AD):#10#3,,#47(07 ) :#247,;
        #200(II SI):#36:#1520#153,.#422(2):#222.;
        #243(AI)D):#16:#IIO.#23(DIUG).#10#3(REGL0REGR):#IOO0#IOI.#36(concat):#IO7;
        #221 (Will IT) ,#10#3(RrGI 0RIGR).#47(O7PC),#23(DREG);
        #200(IEST):#15,,#26(INSIR):#130,#417(BUT37):#217;
L#4:    #360(SIUCT).,#15(DECODF):#141..#775,50.1,#5.0.0.1.#6.1.0.1.#10.2.0,1.#11.3.0.
1,#12.4.0.1.#13.5.0.0.1.#165.6.0.1.#31.7.0.1,#20.8.0.1.#23.9,0.1.#21.10.0.1.#24.11.0.
1.#26.12.0.1.#112.13.0.1,#70.14.0.1.#101,15.0.1.#102,16.0.1.#104.17.0.1.#105.18.0.
1,#114,19,,0,1,#115 .20,,0.1.,#122.21.0,1,#123.22.0.1.#125.23.0.1.#126.24.0.1.#127.25.0.
l.#130.26,,0,1,#154 .27,,0,1.,#131.28.0.1.#136.29.0.1,#777,30.0.1,#134.31.0.1.#777.32.0.
l.#777.33.0.1.#777 .34,,0.1.,#160.35.0.1.#121.36.0.1.#164.37.0.1.#137.38.0.1.#140.39.0.
l.#142.,40.0,1,#153 .41,,0,1.,#143.42.0,1,#144.43.0.1.#145.44,0.1.#146.45.0.1.#147.46.0.
l.#150.,47.0,1,#161 .48,,0,1,#163.49,0;
I#5:    #220(RIAD):#l®#3.,#26(INSIR):#127.;
        #210(MOVI),#25(HARIG),#10#3(RIGI0IULR):#1OO8#1O1.;
        #?20(RIAl)):#l»#3. .#26( INSIR) :#127 . ;
        #200(II SI):#36:#1520#153.,#423(1OR2):#223.;
        #?43(AI)l)):#H):#n0.#23(l)RIG).#18#3(RIGI«RIGR):^1008#101 ,#36(CONCAT ) :#107 ;
        #3O5(JOIN).#7;
I#6:    #220(RIAD):#10#3..#26(1NSTR):#127.;
        #2OO(II.S1):#36:#1520#153,.#423(1OR2):#223,;
        #244(SUH):#16:#UO.#23(1)RFG),#10#3(REGL8REGR):#1OO8#1O1.#36(CONCAT):#1O7;
        #220(RIAD):#10#3.,#26(INSTR):#127.;
        #200(IIST):#36:#1520#153..#423(1OR2):#223.;
        #24 4(SUn):#I6:#1O4.#25(HARIG).#I0#3(RrGI8RrGR):#IOO0#IOI.#36(CONCAT):#IO7;
L#7:    #27 1 (WH1 II ).#1@#3(IUGL0RIGR).^26(1NS1R):#127.#23(I)REG);
        #3b5(JOIN).#15;              !250
L#10:   #22O(RIAD):#10#3.,#26(INSTR):#127.;
        #200(IIST):#36:#1520#153..#422(2):#222.;
        #243(AI)D):#16:#HO.#23(I)IUG).#10#3(REGL0REGR):#1OO8#1O1.#36(CONCAT):#1O7;
        #220(RI AI)):#10#3. ,#26( INSTR) :#127 . ;
        #210(MOVI),#25(HARIG).#10#3(RIGI0RrLR):#IOO0#IOI.;
        #221 (WHI II ).#10#3(RIGI 0RIGR).#26( I NSI R) :#127 .#23(I)IUG);
        #3G5(JOIN).#14;              1245
I#H:    #?20(RI AI)):#10#3. .#26( 1 NSI R ) :#127 . ;
        #?00(II SI):#36:#1G20#153..#422(2):#222.;
        #24 4(SUH)#H>.:#HO.#23(I)RIG).#10#3(RrGI»RIGR):^IOO«#IOI .#36(CONCAI ) :#107 ;
        ff?2Q{\\\ AI)):#1»#3. ,#2G( I NSI R): #127 . ;
        #200( II SI ):#36:#If)20#153. ,#422( 2 )#: #222 . ;
        #244(SUn):#1G:#1O4.#25(IiARIG).#10#3(Rf GI0RIGR):^1OO0#1O1.#36(CONCAI ):#107;
        #721 (Will II ) ,#10#3(RIGI 0RIGR).^2T)( INSI R ) : #127 .#23(I)IU G);
        #3G5(JO1N).#14;  1245
l#12:   #220(RIAD):#10#3..#26(INSIR):#127.;
        #210(MOVI ).#25(HARIG).#10#3(IUGI0RI I R) :#1009#101, ;
        #3G5(JOIN).#15;  1247
L#13:   #220(RI AI)):#1W#3. .#47(07 ) :#247 . ;
        #700(11 SI):#30:#15?»#153.,#422(2):#222.;
```

```
        ^243(add):#16:#110.^23(DREG)v#l@^3(REGL0REGR):#1009#101t#36(CONCAT):#107;
        #221(WRIU).#10#3(REGI0REGR).#47(O7).#23(DREG);
        #220(RtAD).#31(rRtG%0).#2(UNIBUS),#25(BAREG);
        #210(MOVf).#21(HRIG),#31(IIIIG%0):#115.;
        #?20(RI Al)):#10#3.,#2G(INSIN):#127.;
        #243(AI)I)):#16:#11O,#2G(BARIG).#10#3(RIGL0RIGR):#1OO0#1O1.#21(BREG):#1448*145;
        #365(join).#15;          1247
L#165:  #22O(RFAD):#10#3,.#47(07):#247,;
        #200(11 SI):#36:#1520#153.,#422(2):#222.;
        #243(add):#16:#1 10.#23(1)1(1 G).#10#3(lll.GI0KL Gil):#100G#101,#36(CONCAl):#107;
        #221(WRITI).#10#3(RtGI 0RrGR).#47(07).#23(DREG);
        #220(RI AI)).#31(TRIGX0).#2(UNIBUS),#25(BAREG);
        #210(MOVI ).#21(MlLG).#31(IMG%0):#115.;
        #22O(RI Al)):#10#3.,#26(INSTR):#127,;
        #243(ADD):#16:#11O,#25(BARIG),#10#3(RIGL0RIGR):#1OO0#1O1,#21(BREG):#1440#145;
L#14:   #220(111AD).#31(IIIIGZO).#2(UNIBUS),#25(BAREG);
        #210(MOVl),#21(BRIG),#31(MMG%0):#115,;
        #221(write),#10#3(regI0regr),#51(11 source),#31(TRFG%0);
        #22O(RTAI)):#10#3..#51(11source):#251.;
        #210(MOVL).#25(BARlG).#l8#3(RrGL8RFLR):#l008#l0l.;
I#15:   #220(111AD),#31(IHIGXO).#2(UNIBUS).#25(BARLG);
        #210(M0Vl).#21(BRLG).#31(IRLGXO):#115,;
        #221(write).#18#3(regl0regr),#51(11 source).#31(TREGXO);
        #200(HST):#15.,#26(INS1R):#130.#415(BUF35):#215;
L#16:    #360(select).,#15(decode):#157,.#765.16.1.#62.0.0.1.#61,1,0.1.#65.2,0.1,#31,3,0,
1.#20.4.0.1.#23.5.0.1.#21,6.0.1.#24.7.0.1.#2  5.8.0.1,#70,9.0.1.#101,10.0.1.#102.11,0.
1.#104.12.0.l.#105.13.0.1.#106.14.0.1.#17.15.0;
L#17:   #210(MOVI).#23(I)IUG),#21(BRIG):#1448#151;;
        #221(write).#10#3(reg1»regr),#51(11 source),#23(DREG):#116;
        #200(USr):#15.,#26(INSTIt):#130,#416(BUI36):#216;
        #360(SI I IC1), ,#15(1)1CODI):#157, .#755.15.1.#62.0.0.1.#61.1.0.1.#65.2,0.1,#31,3,0,
l,#20.4.0,1,#23.5.0.1.#21.6.0.1.#24.7.0.1,#25.8.0.1.#70.9.0.1.#101,10.0.1.#102.11,0.
l.#104,12.0.1.#105.13.0.1.#106,14.0;
L#20:    #220(RI Al)):#10#3..#26(INSTR):#260.;
        #210(MOVI),#25(BAIUG),#10#3(RLGL0RLLR):#1OO0#1O1(;
        #220(RIAD):#1»#3.,#?6(1NSIR):#260,;
        #200(11 SI):#3(»:#1i2«#103..#423(10R2):#223.;
        #243(AI)l)):#16:#I10.#23(DRIG).#18#3(RrGI8REGR):#1009#101.#36(C0NCA1):#107;
        #365(JOiN),#22;  1260
I#31:   #220(IUAD):#18#3..#26(1NSTR):#260.;
        #210(MOVl ).#25(BARIG),#18#3(RrGI.8RELR):#1008#101.;
        #365(JOIN).#30;  1266
L#21:    #220(RFAD):#18#3,,#26(1NSTR):#260.;
        #200(irSl):#36:#1620#153..#423(10R2):#223.;
        #244(SUB):#16:#11O.#23(l)RIG).#10#3(RFGI0RrGR):#lOO9#10l.#36(CONCA1):#IO7;
        #220(RI Al)):#10#3,,#26(INSIR):#260.;
        #200(11 SI):#36:#1520#153..#423(10R2):#223.;
        #244(SUB):#16:#104,#2r3(BARIG).#l«#3(RIGI 0RI GR):#1009#101.#36(C0NCAI):#107;
L#22:    #221(WRI\V).#10#3(REG10REGR).#26(1NSTR):#260.#23(DREG):#116;
        #365(J01N).#30;  1267
I#23:    #220(RIAD):#10#3.,#2G(INSfR):#260.;
        #210(M0VI).#25(BARIG).#10#3(RIGI0IUIR):#1OO8#1O1.;
        #2?0(RI Al)):#10#3.,#26(INS1R):#260.;
        #200(II SI):#36:#1520#153..#422(2):#222.;
        #243(AI)D):#1(5:#11O.#23(l)RIG).#1»#3(RIGI0RIGR):#1OO0#1O1.#36(concat):#lO7;
        #221(WRIII ).#10#3(RIGI ORIGR).#26(INSIR):#260.#23(DRI G):#116;
        #365(JOIN).#27;  1264
I#24:    #220(RIAD):#10*3,.#2G(INSIR):#260.;
        #200(II SI):#3(5:#1520#153.,#422(2):#222.;
        #244(SUB):#16:#1O4,#2^)(BARIG).#10#3(RIGI0RIGR):#1OO0#1O1,#36(CONCA1):#107;
        #220(III AD):#1003.,#2G(INSIR):#260.;
        #200(11 SI):#3G:#1520#153.,#422(2):#222.;
        #244(SUB):#16:#I1O.#23(DRIG).#10#3(RrCiI0RrGR):#1OO0#1O1.#36(CONCAI):#107;
        #221(WRIII ).#10#3(RIGI0RIGR).#26(INS1R):#260.#23(DRI G):#116;
        #36ft(JOIN).#27;  1264
I#25:    #220(RIAD):#10#3..#47(07):#247.;
```

```
        #210(MOVC).#25(BAREG),#10#3(REGL0REGR):#1000#101,;
L#26:    #220(RFAD):#10#3,,#47(07):#247.;
        #200(HS1):#36:#1520#153..#422(2):#222.;
        #243(ADD):#16:#110,#23(DRIG),#10#3(REGt0IUGR):#1000#101,f#36(concat):#107;
        #221(WRIU).#10#3(RrGL0REGR).#47(07PC).#23(DRrG):#116;
        #220(RrAD).#31(lRrG%0),#2(UNIBUS).#25(BAREG);
        #210(MOVI  ).#21(BREG).#31(1REGX0):#115.;
        ^220(IUAD):^1»/Sf3..#26(INSTR):#260,;
        #243(ADD):#16:#104,#25(HARIG),#10#3(RTGl0RTGR):#1000#101.021(DREG):#1440#145;
        #350(If  ),,#26(1NSIH):#261.,#30.2,#30,#27;
L#27:    #220(READ).#31(TRFGX0).#2(UN1BUS),#25(BAREG);
        #221(WRnE).#18#3(RIGt0RIGH).#52(12DEST).#31(TREGX0);
        #220(Rl.AI)):#10#3..#52(12dest):#252,;
        #210(MOVr),#25(HARrG).#10#3(RIGI8REI R):tf100G/H0l,;
I#30:   #220(Rl.AD),#31(IHIG%0),#2(UNIBUS).#25(BAREG);
        #210(MOVE).#21(BRIG).#31(IRLGX0):#115.;
        #221(WRIir),#10#3(RIGL0R!GR)f#52(12DIST).#31(TREGX0);
        #200(TEST):#15,.#26(lNSTR):#130.#413(BUT33):#213;
        ^360(select).,#15(  decode):#157.,#745,15.1.#33.0.0.1.#35.1,0.1.#40.2,0.1.#41,3,0.
1.#43.4.0.t.#44.5.0,l.#45.6.0.1,#46.7.0.1,#50.8.0.1.#51.9,0.1,#53.10.0.1.#55,11,0,
1.#56.12.0,J,#34,13,0.1,#32,14,0;
L#32:    #210(MOVE):#16.#23(DREG).#21(BREG):#1440#151.;
        #221(WR11I).*10#3(KIGI.0RIGR).#52(12dest):#252,#23(DREG):#116;
        #200(UST):#15.,#26(INSTR):#130.#414(BUI34):#214;
        #360(se1ect)..#15(decode):#157..#735.14,1.#33.0,0.1.#35.1.0.1.#40,2.0,1,#41.3.0,
1,#43.4,0.1.#44.5.0.1.#45,6.0.1.#46.7.O.I.#50.8,0.1.#51.9.0.1,#53.10.0,1,#55,11,0,
l.#56,12.0.1.#34,13,0;
L#33:    #220(RrAD):#10#3.,#52(12dest):#252.;
        #232(decr):#16:#110.#23(dreg).#10#3(regl0regr):#lOO0#lOl.;
        #210(move).#30(pscary),#200#23(10dreg):#1200#117.;
        #365(join),#36;          1367
L#34:    #210(move):#16.#23(dreg).#21(breg):#1500#151,;
        #365(join).#36;          1367
1#35:    #204(neg2c):#16.#23(dreg).#21(breg):#1440#145.;
L#36:    #212(noop);      falter the condition codes
I#37:   #210(move).#31(IRlG7.0),#23(dreg):#114.; (write out the data
        #221(wriI»),#?(unibus).#25(bareg)v#31(IRI'GX0):
        #3G5(join).#121;                 116     this should be a butl6
L#40:    #220(RfAD):#10#3.,#52(12dest):#252.;
        #232(decr):#16:#llO.#23(dreg).#10#3(regl0regr):#lOO0#lOl.;
        #365(join).#42;          !253
1#41:    #254(neg2c):#16.#23(dreg).#21(breg):#1440#145.;
L#42:    #210(move).#30(pscary).#2Off#23(10dreg):#12O0#117,;
        #212(noop);      !set the condition codes sps=l
        #210(M0VI).#21(HRIG).#23(l)inG):#116.;   !SPS=3
        #210(move):#16.#23(dreg).#21(breg):#1500#147,;
        #3<55(join).#37;         1375
L#43:   #220(IUAD):#10#3..#26(INS1R):#127.;
        #235(and):#16:#110.#23(DRrG).#10#3(REGL0REGR):#lOO0#lOl,#21(BREG):#1440#145;
        #365(joln),#36;          1367
I#44:    #220(IUAD):#1R#3,.#51(11source):#251.;
        #23r>(and):#16:#110.#23(ORIG).#10#3(R(Gl0RIGR):#1000#10l.#21(BRrG):#1440#t45;
        #365(join).#36;          1367
I#45:    #220(RIAD):#1G#3.,#26(INS1R):#127,;
        #210(move).#21(breg).#10#3(regI0regr):#121.;
        #30r)(joln),#47;         1365
I#46:    #220(RIAD):#1W#3..#51(11source):#251.;
        #210(move),#21(brey).#10#3(reg!0regr):#121.;
I#47:   #??()(RlAl)):#l»#3..#fi2(12clesl):#252.;
        #244(SUB):#1O:#110.#23(l)RIG).#10#3(IUGL0RIGR):#1OO0#1O].#21(BRIG):#1440#145;
        #3G5([join).#36;         1367
I#50:    #220(IUAD):#1«#3.,#26(INSIR):#127.;
        #235(and):#16:#t1O.#23(l)RIG).#10#3(RrGI0IUGR):#IOO0#1O1.#2I(BRIG):#1440#145;
        #3()f>(joln).#52;        1254
I#51:    #220(RlAD):#1»#3..#51(11source):#261.;
        #235(and):#16:#110.#23(DRIG).#10#3(IUGI0RIGR):#1OO0#IOI.#21(BRIG):#1440#146;
```

```
L*52:   #210(MOVE).#21(BREG).#23(DREG):#116,;     !SPS=1
        #210(MOVt).#21(BREG),#23(DRrG):#116,;     !SPS*3
        #210(move):#16.#23(dreg),#21(breg):#1508#147,;
        #365(join).#37;            1375
1*53:   #210(move):#16,,#21(breg):#1448#145.;
        #?10(inovo).#?0&#?3(l Gdreg).*16(alu):*134.*16(a1u):*110;
        *210(move),*30(pscary),*208*23( l8dreg ):*1208*117.;
        #210(MOVI ).#21(BIUG),*208*23(l 8l)Rf G) :#1208#l 17 . ;
        #210(move):#16.#23(dreg),#21(breg):#1508*151,;
1*54:   *212(noop);      !set the condition codes sps=2
        #365(join).#36;          1367
1*55:   *210(move):*16:*134,*20(l ).*21(breg):*1508*147,;
        #210(movo).#30(pscary),*20«*23(IGdreg):*1208*117,;
        #210(MOVI ), *2 1 (MU G ).*20»*23(l QI)RJG):*1208*117,;
        #210(move):*16.*23(dreg),#21(brec>):*1508*147.;
        *365(join),*54;          1277
1*56:   *211(clear).*23(dreg),,;        !this should be a sign extend
1*57:   *210(move),*30(pscary),*23(dreg):*116.;  Jailer the condition codes .sps=3
        *221(WIUir ),*18*3(IMGI 0RLGN),#26( 1NSIR ):*260,*23(DREG):*il6;
L*60:   #365(join).*1;   !000 this should be a but 27
1*61 :  *220(IU Al)):*1fl*3. .*51 ( 11 source ): *251.;
        #210(move).*21(breg),*18*3(reg18regr):*121.;
        *220(RfAD):*10*3.,*26( JNSIR):*260.;
        *244( SUH):*16:*110.*23(DRfG),*18*3(REGI8RrGR):*1008*101.*21(BREG):*1448*145;
        *365(join),*57; 1360
L*62:   *220(HI AD):*18*3,,*26(INS1R):*26O.;
        *210(move).*21(breg),*18*3(reg18regr):*121,;
        *220(HIAf)):*18*3..*51(11source):*251,;
        *235(and):*16:*110.*23(DREG).*18*3(REGI8REGR):*1008*101.*21(BREG):*1448*145;
L*151:  *200(1tS1):*15.,*26(1NSIR):*130,*411(BUI 31):*211;
        *360(select).,*15(decode):*262..*735.3.1.*57,0.0.1,*63.1.0,1,*64.2,0;
1*63:    *221(write).*l(regl).*26(instr):*260,*23(dreg):*116;
        *365(join).*60;  !to but 27
L*64:   *212(noop);      !alter the condition codes  sps=3
        *365(join).*60;  !to but 27
I *65:   *220(RI AD):*18*3..*51(11 source):*251,;
        *2IO(mov«):*16.*23(dr«g).*18*3(r«gl8regr):*103,;
 1*66:   *210(move):*16,*21(breg),*23(dreg):*116.;
        *221(WRlir).*16*3(RtGI 8RrGR).*52(12DES1)f*23(dreg):#116;
        *2OO(11S1):*15,,*26(INSIR):*130,*400(BUT20):*200;
        *360(select).,*15(decode):*262.,*735.4.1,*t.0.0.1.*133,1.0.1,*134.2,0,1,#67,3,0;
L*67:   *210(move):*16,*23(dreg).*21(breg):*!468*147,;
        *2l0(move):*16.*21(breg).*23(dreg):*116.;
        #221(Will 11 ).*18*3(IUGI 8RIGR).^52(12DrSl ).*23( dreg):#116;
1*152:  *200(II SI ):*15,,*26( INSIR):*130,*407(BUI27):*207 ;
        *360(seIect)..*15(decode):*262..*735.3.l.*1.0.0.1.*133.1.0.1.#134.2.0;
L*70:   #220(RIAD):*18*3..*26(1NSTR):*260,;
        #210(MOVI ).*25(BAIUG).#18*3(REGI 8RFI R) :*1008*101. ;
1*71:    #221(WRITE),*18*3(R[GI8IUGR)t*26(lNSTR):*260.*23(DREG):#116;
L#74:  #200(11ST):*15.,*26( INSIIt):#130.*4O2(BU122):*202;
        #360( select)..#15(decode):*262..#735.4.1.#72.0.0.1.#75.1.0.1.#76.2.0.1.#100.3,0;
L#72:   *220(RIAD):#1S*3.,*51(11 source):*251.;
        *210(move):*16.*23(dreg).*1«*3(regl8regr):*103.;
1*73:   #212(noop);     falter the condition codes  sps=3
        #365(join).#37;
1*75:   #220(111 Al)):#19#3. ,#26( INSIIt):*127, ;
        *200( MSI) :*36:*1528*ir>3. ,*436(00R2) :*236. ;
        *243(AI)I)):*16:*110.*23(DRrG).*18*3(RIGI8IUGR):#1008#101 .#36(CONCA1 ):#107;
        #212(noop);     taller the condition codes  sps=3
        *365(join).*37;
1*76:   *270(RIAD):#18*3.,*51(11 source):*251.;
        #210(move),#21(breg),#1fl*3(reg18regr):*121.;
1*77:    #210(move):#16,#23(drog).*21(breg):*!508*147,;
        *212(noop);     (alter the condition codes  sps=3
        #365(Join).#37;
I#100: *220(RIAl)):*18*3.,*26( INSIR):*127.;
```

```
          *21O(move),*21(breg),*10*3(regl0regr):#121,;
          #365(join),*77;
L*101:    *22O(READ):*10*3.,*26(INSTR):#26O,;
          #210(MOVI).*25(BARfG).*10*3(REGL0RELR):*lOO0#lOl,;
          #22O(READ):*10*3. ,*26( INSFR) :*260 . ;
          *200(TEST):*36:#1520*153. .#4?3( 1OI?2)-#7?3. ;
          #243(AOD):*16:*llO.*23(DRLG).*1»#3(RIGI0RIGR):*lOO0*lOIf*36(CONCAT):*lO7^
          *365(jo1n),*71;
1*102:    #220(Rf AI)):*10#3. ,*26( INSIR) :*260. ;
          *210(MOVI ).*25(BAHIG).*1»#3(RIGI Hill I II) : 01000*101 , ;
          #220(READ):#10*3.,*26(INSTR):*260.;
          *2OO(IES1):*36:*1520*153..*422(2):*222.;
          *243(ADI)):*16:*nO.*23(I)IUG).*10*3(RIGI0RIGR):*lOO0*lOI,*36(concat):#lO7;
          *221(WRMI ) ,#10*3(111 GI 0RIGR).*26( INS1R):#260,*23(DREG):*116;
L*103:    *220(Rf AU),*31( I Rf G7.0 ) ,*2(UN1BUS), *25(BARFG);
          #2 10(MOV! ) ,*2 1(Dili G ) .*31 ( 1 RE GXO): *115, ;
          *22 1 (Will IE ) . * 10*3 (III GL0RIGR),*52(12DES1),*31(IREG%0);
          *220(READ):#10*3.,*52(12desl):*252.;
          *210(M0VI).#25(BAREG),*10*3(REGL0RELR):*1OO0*1O1,;
          *365(join).*74;
1*104:    *220(RIAO):*10*3, ,*26( INSIR ) :*260 , ;
          *200(HSF):*36:*1520*153. ,*423( 10R2 ) :*223 . ;
          *244(SUH):*16:*1O4.*25(HARrG).*10*3(REGL0REGR):*lOO0*lOIv*36(CONCAT):*lO7;
          *22O(RrAD):*10*3.,*26(INSIR):*260,;
          *200(TESI):*36:*1520*153..*423(10R2):*223,;
          *244(SUB):*16:*llO.*23(DRIG).*10*3(RtGL0RIGR):*lOO0*lOI(*36(CONCAT):*lO7;
          *365(join),*71;
L*105:    *22O(RIAD):*10*3..*26(INSIR):*260,;
          *200(TFST):*36:*l520*153,t*422(2):*222,;
          *244(SUB):*16:*1O4.*25(BAR[G).*10*3(REGL0RFGR):*1OO0*1O1,*36(CONCAT):*1O7;
          *22O(RrAI)):*10*3. .*26( 1NS1 R): *260 , ;
          *200(UST):*36:*1520*1 53. .*422( 2 ) :*222 . ;
          *244( SUB ): *16 : *110. *23( Dili G).*10*3( III GL0RIGR):*! 000*101 t*36(C0NCAT): #107;
          #221(WfUTL).*10*3(IUGL0RLGR).*26(INS1R):*26O.*23(DRr.G):*116;
          *3(55(join).*103;
1*106:    *22O(RrAI)):*10*3. ,*47( 07 ) :*247 , ;
          *2 1()(MOVI ).*2r>(BAIIIG).*10*3(RIGI0RIGR):*1000*101. ;
          *220(III AI)):*10*3, ,*4 7( 07 ) :*247 . ;
          *200(rFST):*36:*1520*153.,*422(2):*222,;
          *243( ADO):*16 : *110.*23(DiliG ) ,*10*3( REGL0RTGR ) :*l000*101 ,*36(concat):#107;
          *221(WRITt ).*10*3(RIGL0RrGR).*4 7(O7PC).*23(DREG):*116;
L*107:    *220(RLAI)).*31( I Rf GXO) .*2( UNI BUS) .*25(BAREG);
          *210(MOVE),#21(BRE G),*31(1REGXO):*115,;
          *221(WRI1I),*10*3(RIG10RIGR),*52(12DfST),*31(IREGXO);
          #200(US1 ):*15. ,#26( I NSIII) :*130 ,*77(BU1 1 7 ) :*177 ;
          #360(select),,*15(decode):*263.,#735.2.1.#110,0,0.1.#111,1,0;
L#110:    *22O(IUAI)):*10*3..*26(INS1R):*26O.;
          *243(ADD):*16:*1O4.*25(BAIUG),*10*3(RFGE0IUGR):*1000*101,*21(BREG):*1440*145;
          #200(TEST):*15..*26(INSFR):*130.*401(BUT21):*201;
          *360(SGIect)..#16(decode):#262,.#735.4.1.#72.0.0.1,#75.1,0,I,#76.2.0.1,#100,3,0;
1*111 :   #220(RI AD): #10*3. ,*2f>( INS1 R ) :*260 . ;
          *243( ADD ):*16:*1O4.*25(BAIUG),*10*3( III GI 0111 GR ) :*1OO0*101 ,*2 1 (BRI G) :*1440*145 ;
          #365(join).*103;
1*112:    *220(RI AI)):*10*3. , *26( 1 NS1 R ) : *127 . ;
          *21()(move):*16.#23(dreg).*10*3(regl0regr):*IO3.;
          *365(join),*66;
1*114:    *22()(RI AI)):*10*3, .*47( 07 ) : *247 . ;
          #200( II SI ): #3(5 :*1520*1 53. .*422( 2 ) : *222 . ;
          #243( add ) :*I6 :*11O.*23(DIIIG).*10*3( III GI 0111 GR):*1000*101 ,*36(C0NCAT ) :*107 ;
          #221(WIIIM ).#10#3(RIG1 0RIGII).#47(O7).#23(DIILG):#116;
          *365(join).#107;
I#115:    *22()(RI AI)):*10*3, ,*26( 1 NS1 R) :*260. ;
          #210(M0VI ):*16:#11O.*23(I)RIG).*10*3(RIG10RH R) :#1OO0#1O1, ;
I #116:   #21()(MOVI ).#? 1(IIIIIG).#23( 1)111 G):#1 16. ;
          #721 (Will II ).# 1(4*3 (III GI 0RIGR).*5O( 10lemp ) .*23(I)IU G) :*116;
I *113:   *?()()( II SI ):*1f>. .*?()( INSIII):#I3O.#75(IIUI 15):*17[];
```

```
        #360(select).,#15(decode):#263,.#735,2,1,#120.0,0.l.#l17,1,0;
L#117:  #22O(READ):#10#3..#46(O6sp):#246.;
        #2OO(11S1):#36:#1520#153..#422(2):#222.;
        #244(sub):#16:#llO.#23(l)RFG)ₜ#10#3(REGL0REGR):*lOO0*1Ol,*36(CONCAT):*lO7;
        #22O(READ):#10#3.,#46(O6sp):#246,;
        #200(US1 ):#3fi:#15?0#153, ,#422(2):#222, ;
        #244(SUIO:#16:#1O4.#25(HAIUG).#10#3(IUGI0HIGR):#1OO0#1O1.#36(CONCAT):#1O7;
        #221(WRIir),#10#3(IUGI 0RI GR)ₜ#46(06sp),#23(DRIG):#116;
        #22O(IUAD):#10#3, ,#26( INSIR) :#127 , ;
        #21O(n!Ove):#16,#23(dreg).#18#3(reg10regr):#IO3,;
        #210(move)₍#31(1RFG^X.0).#23(dreg):#114₍; lwrite out the data
        #221(write).#2(unibus),#25(bareg).#31(IRtGXO);
        #22O(RLAI)):#10#3, .#47( 07 ):#247, ;
        #21O(movf»),#21(breg).#10#3(regl0regr):#121,;
        #210(move):#16.#23(dreg),#21(breg):#1460#147,;
        #2?1(WR]IT),#10#3(RIGL0RIGH).#26(1NS1R):#127.#23(DREG):#116;
L#120:  #22O(READ):#10#3. ,#50( 10temp): #250, ;
        #21O(move):#16.#23(dreg).#10#3(reg10regr):#lO3,;
        #221(WR111),#10#3(RtGI0RIGR),#47(07pc).#23(DREG) :#116;
L#121:  #200(11 SI):#15.,#26(INS1R):#130.#76(DU!16):#176;
        #360(select)..#15(decode):#263..#735.2.1.#133.0.0,1.#134,1,0;
I#122:  #220(RTAl)):#10#3. ,#26(INSTR):#260. ;
        #200(n SI):#36:#1520*153.,#422(2):#222.;
        #243(AI)l)):#16:#11Oₜ#23(DRFG).#l0#3(REGI0REGR):#lOO0#1O1 ,#36( concat) :'#107 ;
        #221(WIUH ) ,#10*3(HLGI 0IUGR).#26( INSIR) :#260 .#23(DREG): #116;
        #22O(RIAD):#10#3. ,#26( INSIR):*260. ;
        #200(HS? ):#36:#1520#153, ,#422( 2 ) : #222 . ;
        #244(SUH):#16:#11O.#23(DREG).#10#3(REGL0REGR):#1OO0#1O1.#36(CONCAT):#1O7;
        #365(join).#116;
I#123:  #2?0(RI Al)):#l»#3. ,#26( INSTR):#260 . ;
        #210(MOVI ).#25(I3ARIG).#10#3(IUGLGRI I R) :#1009#101, ;
        #220(RLAI)):#18#3. .#26( INSIR) :#260. ;
        #200(11 SI):#36:#1528#153..#422(2):#222.;
        #243( ADD) :# 16 : #110 ,#23(I)IU G ). #10#3( IUGL8REGR ): #100G#101 ,#36( concat): #107;
        #221 (WIU II ) .#1®#3(RIGIORIGR).#26( INS1 R) :#260 .#23(DIUG) :#116;
I#124:  #220(RI AD).#31(TIUGZ0).#2(UNIHUS),#2 5(BAREG);
        #? 10(MOVI ).#2I(HIMG).#31(IRICX0):#115.;
        #221(write).#10#3(regl0regr).#5O(IOtemp).#31(IREG%O);
        #365(join).#113;
L#125:  #22O(RrAD):#10#3..#26(INSTR):#260.;
        #200(USI ):#36:#1520#153. .#422( 2 ) :#222 . ;
        #244(SUH):#16:#11O.#23(I)REG).#10#3(R(GI0REGR):#1OO0#1O1,#36(C0NCAF):#107;
        #221(WRI1E).#1G#3(RLG10RIGR).#26(INSIR):#260,#23(DRLG):#116;
        #365(join).#116;
I#126:  #220(IU AI)):#1W#3. ,#26( INSIR) :#260. ;
        #200( 11 SI):#36:#1520#153.,#422(2):#222.;
        #244(SUH) :#16:#104,#25(HAREG) .#10#3(REGI.0REGR):#1009#101.#36(CONCAT):#107;
        #220(RIAl)):#10#3. ,#26( INSIR) :#260. ;
        #200(TCSI):#36:#1520#153.,#422(2):#222.;
        #244(SUH):#1O:#11O.#23(I)RIG).#10#3(RIGI0IUGR):#1OO0#1O1.#36(CONCAT):#1O7;
        #221(WRI II ).#10#3(IUGI0RIGR).#26( INSIR ) : #260 .#23(I)RI G ) : #116;
        #365(join).#124;
I#127:  #220(RI Al)):#10#3. .#47( 07 ) :#247 . ;
        #200( MSI):#36:#1520#153..#422(2):#222.;
        #243(add):#16:#11O.#23(l)RIG).#10#3(RIGI0RIGR):#lOO0#lO1 .#36(CONCAl ):#107;
        #221 (WRI II ) ₜ#1(4#3(III GI 8RIGR). #47(07) ,#23(I)RI G ) : #116;
        -#220(1(1 AD). #31 ( IRIGX0) ,#?(UN IHUS ) ,#25(HAREG ) ;
        #210(MOVI ).#21(HRIG).#31(IRIGZ0):#115,;
        #??1(WIIMI ) .#10#3(RIGI 8RIGR).#50( 10 temp ) .#31 (1 RTGXO) :#116 ;
        #220(RI AI)):#1W#3, ,#26( INSIR) :#260. ;
        #243(add):#16:#1 10 .#23(1)111 G ) .#10#3( REGI 0Rt GR ): #1OO0#1O1 .#2 1 (Bill G) : #1440#145;
        #365(join).#116;
I#130:  #220(RI AD):#10#3.,#47(07):#247.;
        #200(11 SI):#36:01520*153..#427(2):#?22.;
        #243(add):#16:#11Oₜ#23(l)Rir.)ₜ#10#3(RIGI0RIGR):#1OO0#1O1 .#36(CONCAI ):#107;
        #2? 1(WRI II ) .#19*3(1(1 GI 0KIGR).#4 7(O7).#23(!)RIG):#11(5;
```

```
          #220(READ),#31(TREGX0).#2(UNIBUS),#25(BAREG);
          #210(MOVE).#21(BREG).#31(TREG%0):#115.;
          #221(WRIU).#18#3(REGI8REGR).#50(10temp).#31(TREG%0):#115;
          #220(RFAD):#18*3.,#26(1NS1R):#260,;
          #243(add):*]6:0110.#23(DREG),#18#3(REGL8REGR):#1008#101,#21(BREG):#1448*146;
          #220(RFAD):#18#3. .#26( 1NSFR): #260. ;
          #243(ADD):#16:#104.#25(BARIG),#18#3(RI GL8RI.GR):#1008*101.#21(BREG):#1448*145;
          #365(join),#124;
L*131:     #220(READ):#18#3,.#26(INSFR):#260,;
          #210(MOVF ):#16:#110.#23(DREG).#18#3(REGL8RnR):#1008#101.;
          #221(WRITE).#18#3(REGI 8REGR),*47(07pc),*23(DREG):#116;
          #220(IUAD):#18#3,,#46(06sp):#246.;
          #210(MOVI),#25(HARLG).#18#3(RIGI8REGR):#1008#101.;
          #22O(RFAD):#1G#3,.*46(06sp):*246.;
          #200(ItST):#36:#1528*153.,#422(2):#222,;
          #243(ADD):#16:#110.*23(DRLG).*18#3(REGl8REGR):#1008#101,#36(concat):#107;
          #221(WIUU).#18#3(RIGI 8RIGR).#46(06sp).#23( DREG): #116;
          #220(IU AD),#31(TRIGZ0).#2(UNI BUS),#25(BAREG);
          #?21(WRT1F ),#18#3(IUGI 8RFGR).#26(INSTR):#260,#31(TREGXO):#115;
          #365(join).#121;
I#132: #220(MAl)):#18#3.,#47(07):#247,;
          #210(MOVE).#25(»ARr:G).#18#3(REGL8REGR):#1008#101.;
          #3G5(join),#133;
I#134:       #2OO(1FST):#15..#26(JNSfR):#130,#406(BUT26):#206;
          #360(select).,*15(decode):*262..#735.4,1.#777.0.0.1.#777.1.0.1,#777,2,0,1.#135,3,0;
L#135:     #220(HFAD):#18*3..#47(07):#247.;
          #210(MOVI ).#25(»AREG).#18#3(RrGL8REGR):#1008#101,;
          #365(join).#133;
L#136:    #220(IU AD):#18#3.,#46(06sp):#246.;
          #210(MOVf).#25(UAIUG).#18#3(RfGI8REGR):#1008#101,;
          #220(RI Al)):#18#3. .#46( 06sp ): #246, ;
          #200(IlST):#36:#1528#153,.#422(2):#222.;
          #243(ADD):#16:#110.#23(DIUG).#18#3(RIGI 8RI GR):#1008#101,#36(concal):#107;
          #221 (WR1 FT) .#1Q*3(HI.GI 8RFGR ) .#46( 06sp ) .#23(DREG ) :#116;
          #220(111 AD). #31 ( IltI GXO ) .#2(UN I BUS) .#25(BARFG) ;
          #221(WIUir),#18#3(RIGI 8RI GR),#47(07PC).#31(1REGXO):#115;
          #?20(RI AD) :I/1«#,?. .#4(>( ()(>sp ): #246. ;
          #210(M0VI ).#2r)(liARIG).#18#3(IUGI8REGR):#1008#101. ;
          #220(IU AD):#18#3..#46(06sp):#246.;
          #2OO(1FS1):#36:#1528#153.,#422(2):#222.;
          #243(ADD):#16:#110.#23(DRIG).#18#3(RFGI 8REGR):#1008#101,#36(concat):#107;
          #221(WR!lb).#l«#3(RIGI 8IUGR).#46(06sp),#23(DRIG):#116;
          #220(RFAD).#31(IRIGXO).#2(UN I BUS),#25(BAREG);
          #210(move).#3()«#3?8#338#34(psword).#31( IRIGXO) :*116. ;
          #365(join).#134;
L#13 7:   #220(111 AD):#18#3. .#4 7( 07) :#247. ;
          #243(ADD):#16:#U0.#23(DREG).#18#3(REGF8REGR):#1008#101.#21(BREG):#1468#147;
          #221(WRHt) ,#18#3(IUGI 8REGR) ,#4 7( 07pc ) .#23(DRIG) :#116;
          #220(RF.AD):#18#3. ,#47( 07 ) :#247. ;
          #243(ADD):#U>:#110.#23(DRFG).#18#3(RIGI8IUGR):#1008#101 .#21 (ItRI G ): #1468#147 ;
          #221 (WRI II ).#I8#3(R(GI 8RIGR).#4/(07pc).#23(I)RIG):#116;
          #365(join).#121;
L#140:    #220(RI AD):#18#3..#26( 1NSIR):#12 7.;
          #200( II SI ):#36:#1528#153..#421(1):#22I.;
          #244(sub):#10:#110.#23(I)RIG).#1«#3(RIGI8RIGR):#1008#101,#36(concal):#107;
          #221 (WIU II ).#1ti#3(KIGI 8RIGR).#20( 1 NSIR ) :#127 ,#23(DRI.G);
          #220(RI Al)):#18#3. ,#55( 13): #255. ;
          #200(11 SI ).#36:#1528#153. ,#433( shjc13 ) :#233 . ;
          #235(Anl)):#1O:#1IO.*23(DRIG).*18*3(RIGI8RIGR):#1OO8#1O1 ,#36(C0NCA1 ):#107;
          #210(MOVI ).#21(BRIG),#23(I)IUG):#116.;    ISPS=3
          #200(MSI):#15,.#26( INS IR):#130.#72(BUI12):#172;
          #300(select).,*H>(decode):*263.,#735.2.1.#141.0.0.1.#121.1,0;
I#141:    #220(RI AD):#18#3;.#4 7(07):#24 7,;
          #244(sub):#16:#110,#23(DRIG).#18#3(RIGI8IUGR):^1008#10l.#21(HR! G):#144fl#146;
          #221(WIU1I ).#18#3(RIGI 8RI GR ) .#4 7( 07pc ) .#23(DRrG) :#116;
          #22O(RIAD):#18#3..#4 7(07 ) :#247.;
```

```
        #244(sub):#16:#110,#23(DREG) .#18#3(REGL8REGR):#1008#101,#21(BREG):#1449*146;
        #221(WRITE),#18#3(REGL8REGR),#47(07pc),#23(DREG):#116;
        #365(join).#121;
L#142:  #220(RIAD):#18#3. .#53( 13instr) :#253 . ;
        #200(UST):#36:#1528#153.,#432(sbcl2):#232,;
        #235(AND):#16:#110.#23(DIUG).#18#3(RrGI8RrGR):#1008#101.#36(concat):#107;
        #21O(MOVI ),#21(BRIG).#23(DRlG):#116, ;
        #230(not):#16.#23(dreg),#2l(breg):#1448*145,;
        #21O(MOVt),#21(BRLG).#23(DREG):#116.;
        #235(AND):#16:#110.#23(DRrG),##308#328#338#34(status):#274,#21(breg):#1448#145;
        #210(move).#308#328#338#34(psword).#23(dreg):#116,;
        #365(join).#121;
I#143:  #220(111.AD):#19*3,,#53( 13instr):#253 . ;
        #200(fISl):#36:#1528#153.,#432(sbcl2):#232,;
        #235(AND):#16:#110.#23(DRrG).#18#3(RrGL8REGR):#1008#101,#36(concat):#107;
        #210(MOVI ).#21(BRrG),#23(DREG):#116, ;
        #236(OH):#16:#110,#23(DHLG).##308#328#338#34(status):#274,#21(breg):#1448#145;
        #710(move).#308#328#338#34(psword).#23(dreg):#116',;
        #365(join).#121;
L#144:  #220(RI Al)):#18#3,,#26( INSTR) :#260 . ;
        #21O(movG).#21(breg),#10#3(reg18regr):#121.;
        #22O(RrAI)):#10#3,,#26( 1NSIR) :#127,;
        #243(A()[]):#16:#110.#23(DR(G)>#18#3(RLGL8RLGR):#]008#101,#21(BREG):#1448#145;
        !this operation is a function of the Instr reg
        #365(join).#151;         !to but31
I#145:  #220(RI Al)):#10#3. ,#26( INS1R) :#127 . ;
        #210(move).#21(breg).#10#3(reg18regr):#121,;
        #22O(RLAI)):#10#3. ,#26( 1NS1R):#26O, ;
        #244(SUB):#16:#110.#23(DRrG).#18#3(REGL8REGR):#1008#101.#21(BREG):#1448#145;
        #365(join).#152;         !to but27
I#146:  #22O(RIAI)):#10#3,,#26( INSIR):#260 . ;
        #2l0(move).#2I(breg).#18#3(regl8regr):#121.;
        #210(move):#16.#23(dreg).#21(breg):#1508#151.;
        #365(join).#152;         !to but27
L#147:  #220(Rl AD):#1«#3..#26(1NS1R):#260,;
        #232(decr):#16:#110.#23(dreg).#18#3(regl8regr):#1008#101,;
        !this operation should be a function of the instr
        #210(move),#30(pscary).#23(dreg):#116,;
        #365(join).#151;         !to but31
I#150:  #220(RI AD):#18#3.,#26(INSTR):#260.;
        #210(move).#21(breg).#18#3(reg18regr):#121,;
        #254(neg2c):#16:#110.#23(dreg).#18#3(regl9regr):#1008#101(;
        #365(join),#151;         !to but31
1#153:  #211(c1ear).#23(dreg),,;          !th1s should be a sign extend
        #365(join).#57;
L#154:   #200(HST):#36:#1529#153. .#420( sbcOO ) :#220, ;
        #210(M0VI.):#16:#110.#23(DRtG).#36(concat):#107.;
        #221(WRIII).#18#3(RLGIfIREGR),#54(14vect),#23(DRFG):#116;
I#155:    #220(RrAD):#18#3..#54(14vect):#254,;
        #200(TfSr):#36:#152»#153..#422(2):#222.;
        #243(AI)D):#16:#1O4.#25(BAIUG).#10#3(RIGI8IUGR):#1OO8#1O1(#36(CONCAT):#1O7;
I.#156:   #220(HI Al)).#31( I RI GX0). #2( UNIBUS) ,#25(BARIG);
        #221 (WRI II ) .#1ft#3(RIGI (9HIGR).#50( 10temp) ,#31 ( IRf GX0);
        #220(RI Al)):#10#3. . #46( 06sp): #246. ;
        #200(II   SI):#36:#152W#153..#422(2):#222.;
        #244( sub): #16: #110,#23(DRIG).#18#3(RIGl 8RI.GR ):#1008#101.#36(CONCAT):#107;
        #220(111Al)):#1«#3.,#46( 06sp):#246.;
        #200(11 SI):#36:#152fl#153..#422(2):#222.;
        #?44(siih):#U):#1O4.#75(HARIG).#10#3(RIGI8IUGR):#lOO8#1O1 .#36(CONCA1 ) :#107 ;
        #721 (Will II ) .#IH#3(KIG1 Hill Gil ).#46(0Gsp).#23(l)RLG);
        #210(mov«).#23(drog).#3O«#320#330#34(psword):#2  74.;
        #210(move).#31( I RlG'/.O ) .#23( dreg ): #1 14 . ; Iwrlte out the data
        #221(write),#2(unIbus).#25(bareg),#31(IREGXO);
        #212(noop);      !mm=04
        #220(RIAl)):#10#3. .#46( 06sp):#246.;
        #200( II SI ):#36:#1520#153..#422(2):#222,;
```

```
          #244(sub):#16:#110,#23(DREG).#10#3(REGL0REGR):#1000*101.#36(CONCAT):#107;
          #22O(READ):#10#3.t#46(06sp):#246,;
          #2OO(TES1):#36:#1520#153..#422(2):#222,;
          #244(sub):#16:#lO4,#25(BARrG).#10#3(REGI0REGR):#lOO0#lOl.#36(CONCAT):#lO7;
          #221(WRlU),#10#3(REGL0REGR).#46(O6sp).#23(DREG);
          #220(RIAD):#10#3.,#47(07):#247,;
          #210(MOVl),#23(DRIG).#10#3(RIGI0REGR):#1OO0#1O1.;
          #210(move).#31(TRIG*.0)t#23(dreg):#114, ;  IwrUe out the data
          #221(write),#2(unibus),#25(bareg).#31(TREGX0);
          #22O(RFAD):#10#3,,#50(10temp):#250.;
          #21O(move).#3O0#320#330#34(psword).#10#3(regl0regr):#lOO0#lOl.;     !$ps=0
          #22O(RLAD):#10#3. .#50(10temp):#250.;
          #21O(move).#3OO#320#330#34(psword).#l0#3(reg10regr):#lOO0#lOl.;     !sps=7
I#157:    #220(RIAD):#10#3.,#54(14vect):#254,;
          #210(MOVE).#25(DARI G). #10#3( IUGI.0REGR):#1OO0#1O1,;
          #220(RKAD),#31(IRIGXO),#2(UNI BUS),#25(BAREG);
          #221(WRITE)t#10#3(RtGL0RtGR),#47(O7pc).#31(TIUGXO);
          #200(TFS1):#15..#26(INSTR):#130.#61(BUr01):#161;
          #360(select),.#15(decode):#263.,#735,2,1.#77  7,0.0,1,#134,1,0;
L#160:    #2OO(IEST):#36:#1520#153.,#420(sbcOO):#220,;
          #210(MOVE):#l6:#110.#23(DRIG)t#36(concat):#l07,;
          #221(WRIlE),#10#3(REGL0RrGR).#54(14vect).#23(DREG):#116;
          #365(join),#155;
L#161:    #22O(RrAD):#10#3,,#26(INSTR):#260.;
          #210(move):#16,,#10#3(regl0regr):#1OO0#1O1.;
          #2lO(move).#2O0#23(l0dreg)t#16(a1u):#l34.#16(a1u):#llO;
          #210(move).#30(pscary),#2O0#23(10dreg):#12O0#117,;
          #210(MOVt ).#21(BRLG),#2O0#23(10I)REG):#12O0#117. ;
          #21O(move):#16.#23(dreg).#21(breg):#15O0#151t;
          #221(WRITr).#10#3(RIGI 0RfGR).#26(INSTR):#260,#23(DREG):#116;
L#162:    #212(noop);      !aller the condition codes  sps52
          #365(join).#152:         !to but27
L#163:    #22O(RtAD):#10#3,.#26(INSTR):#26O,;
          #21O(MOVE).#21(BRIG).#10#3(RIGL0RELR):#1OO0#1O1,;
          #210(move):#16.,#21(breg):#15O0#147,;
          #21O(inove).#2O0#23( I0dreg),#16( alu) :#134,#16( alu) :#110;
          #210(move).#30(pscary).#23(dreg):#116.;
          #210(MOVI ).#21(HniG).#2O0#23(lffl)RrG):#12O0#117.;
          #21O(move):#16.#23(dreg),#21(brog):#1440#151.;
          #221(WR]1f ),#10#3(RCGL0R(GR).#26(INSIR):#260,#23(DREG):#116;
          #365(jo in),#162;
L#164:    #220(RIAD):#10#3,,#53(13instr):#253,;
          #243(AI)D):#16:#nO.#23(l)REG).#10#3(REGL0REGR):#lOO0#lOl.#21(BREG):#1440#145;
          #210(MOVr ).#21(BRIG).#23(l)RlG):#116f ;
          #220(Rl AD):#10#3,,#47(07):#247.;
          #243(ADD):#16:#nO,#23(DRFG).#10#3(REGL0REGR):#lOO0#lOl,#21(BREG):#1460#147;
          #22O(READ):#10#3,,#47(07):#247,;
          #243(ADD):#16:#1O4.#25(BARIG).#10#3(REGI0REGR):#1OO0#1O1.#21(BREG):#1460#147;
          #221(WRnr).#10#3(REGL0REGR).#47(O7pc).#23(DRtG):#116;
          #220(RIAl)):#10#3. .#47(07):#247.;
          #200(11 SI):#36:#1520#153.,#422(2):#222.;
          #243( ADD) :#1(i: #110. #23(0111 G).#10#3(RIGI 0RI GR ) :#100»#101 ,#36(CONCAI ):#107;
          #221(WRIII ).#10#3(IUGI 0RIGR).#46(O6sp).#23(DRIG):#116;
          #220(Rl AD):#10#3..#45(05R5):#245.;
          #210(move):#1O.#23(dreg),#10#3(regI0regr):#103.;
          #220(RI Al)).#31( Mil CX0 ) ,#2(UNIBUS) .#25(BARrG) ;
          #221(WIN II ).#1G#3(RIGI GRIGR).#45(06R5).#31(IRIGXO);
          #221(WRIII).#10#3(RIGI0IUGR).#47(O7pc).#23(dreg):#116;
          #365(joln).#121;
I#775:    #373(SMIRGI).#4;
L#777     #351(l»l Nl)) .#1(Pl)l>1140);
```