Chapter 5

Microprogramming

A processor is composed of a data path and a control unit. The data path of a processor consists of execution units, such as an ALU, a shifter, registers, and their interconnects. A control unit is considered to be the most complex part of a processor. Its function is to control various units in the data path. The control unit realises the behaviour of a processor as specified by its instruction set. The performance of a control unit is crucial as it determines the clock cycle of the processor.

A control unit can be implemented in either hardwired or microprogram. A hardwired control unit is a large FSM (finite state machine) sending control signals to the data path. A microprogrammed control unit [FLY71] is a complex programmable unit that outputs control signals to a data path according to its *microprogram*. A microprogrammed control unit can be regarded as a simple computer. In this view, a processor has another simple processor inside it which is its control unit. Controlling a data path is represented by its microprogram. We will discuss microprogramming concept in details in this chapter. It will be used in the next chapter to design the control unit for the processor used in our system.

5.1 Hardwired control unit

In the past, a hardwired control unit was very difficult to design hence its engineering cost was very high. Presently, the emphasis of computer design is the performance therefore hardwired is the choice. Also the CAD tools for logic design have improved to the point that a complex design can be mostly automated. Therefore almost all processors of today use hardwired control units.

Starting with a behavioural description of the control unit, a state diagram of micro-operations is constructed. Most states are simply driven by clock and only

transition to the next state. Some states branches to different states depended on conditions such as testing conditional codes or decoding the opcode.
From the state diagram, a hardware realization can be constructed almost automatically by some CAD tools. Explanation of logic design for sequential circuits and logic minimization can be consulted from many basic textbooks on the subject such as Katz [KAT96]. The control circuit is implemented using Programmable Logic Array (PLA). In general, any sequential circuit (which can implement any state machine) can be constructed from a combinational circuit with feedback. The feedback signals represent the states. If the feedback path uses no clock, the circuit is called *asynchronous*. If the feedback path uses a latch with clock, the circuit is called *synchronous*. Synchronous circuits are used almost exclusively for sequential circuits today as they are easier to design and can be implemented reliably. Most of the CAD tools handle synchronous circuits. Asynchronous circuit has been used for the reason of performance as in many early computer designs, for example, ILLIAC and many computers in the class called supercomputer. But it is difficult to implement reliably and it is still much more difficult to do systematic design of a complex machine using asynchronous circuits. The combinational part of the control circuit can be regarded as a memory where its content is the map of the input to the output (states are considered to be a part of the output). This view of combination circuit as a memory is called Random Access Memory model (RAM) of computation machines.

5.2 Microprogrammed control unit

Maurice Wilkes invented microprogram in 1953 [WIL53] [WIL85]. He realised an idea that made a control unit easier to design and made it more flexible. His idea was that a control unit can be implemented as a memory which contains patterns of the control bits and part of the flow control for sequencing those patterns. A microprogram control unit is actually like a miniature computer which can be programmed to sequence the patterns of control bits. Its “program” is called microprogram to distinguish it from an ordinary computer program. Using microprogram, a control unit can be implemented for a complex instruction set which is very difficult to do by hardwired.

How microprogram work

The simplest way to understand a microprogrammed control unit is to regard it as a ROM which implements a finite state machine (FSM). A general sequential circuit consists of a combinational circuit which some output is fed back to input. If the feedback part is synchronised with a clock using a register to latch the signal, then the circuit is said to be *synchronous*. That is, the output changes at the edge of the clock, between the clocks, the output does not change (Fig. 5.1). If the output is fed back directly, it is said to be *asynchronous*, the output changes dependent on the delay of the combinational part and it can change ("oscillate") a number of times before the signal is settled (or never settled). This is the most general way to visualise a sequential circuit. Any combinational circuit can be directly implemented as a truth table, although it is not efficient in terms of size. A read-only-memory (ROM) stored the truth table. In a sense, this ROM can be regarded as a "program".

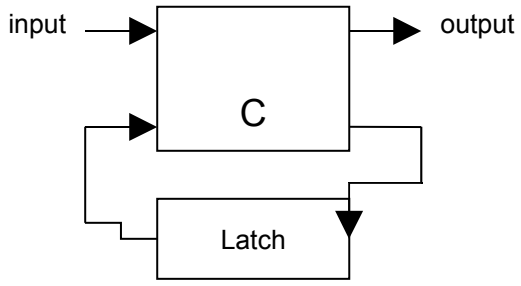


Figure 5.1 A general sequential circuit

A control unit outputs the control signals to control various parts of the data path such as selecting a multiplexer, latching a register, or controlling the operation of an ALU. The first stage to see how a ROM can be used as a control unit is that the ROM can output a fixed sequence of control signals simply by cycling the address of the ROM. The content of this ROM is a microprogram, but it is comparable to a *straight line* program, i.e. the one without any transfer of control (Fig. 5.2). This is how the inventor of microprogram, Maurice Wilkes, discovered it. We will call each entry in this ROM a *microword*.

A microprogram counter is used to cycling the sequence of control. If a part of ROM, says a number of the right most bits, is used as the next address, then these bits can be loaded into the microprogram counter to alter the sequence of control. We now have a microprogram with "goto". Some control bits can be used to test

the condition, such as the zero flag. The result of testing can be used to decide whether to load the microprogram counter or to increment it. This is equivalent to the statement "if..then". Changing the behaviour of the control unit can be done simply by changing this microprogram (Fig. 5.3).

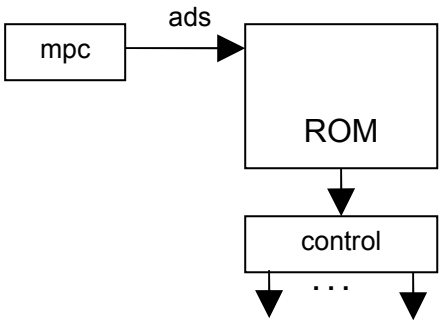


Figure 5.2 A fixed sequence control unit using ROM

Conditionals are the bits that are used to determine the flow of microprogram; loop, branching, next instruction etc. Its input comes from the data path (usually from the conditional code register). The next address determines the next microword to be executed.

A microprogram is executed as follows.
1. A microword at the location specified by the microprogram counter is read out; control bits are latched at an output buffer which is connected to the data path.
2. If the conditional field is specified and the test for conditional is true, the next address of microprogram will come from the next address field otherwise the microprogram counter will be incremented (execute the next microword).

What that has been described is called *horizontal microprogram*. The microword can have other formats. There are several possibilities (Fig. 5.4):
1. Single format, one address as just described above.
2. Single format, two addresses, contain two next addresses field, one for result of test true, and the other for result of test false.
3. Multiple formats, such as, one format for the control bits without the next address field and another format for *jump on condition* with the address field.

The advantage is that the microword can be shorter than the single format. The disadvantage is that to "jump" will take one extra cycle.

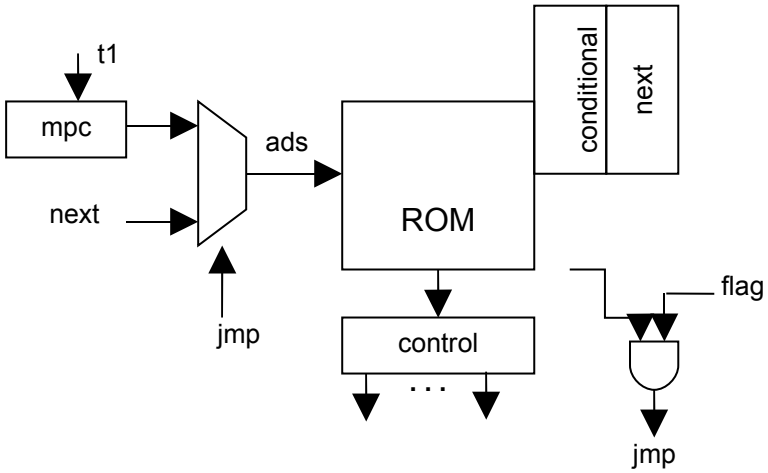


Figure 5.3 A fully function microprogrammed control unit

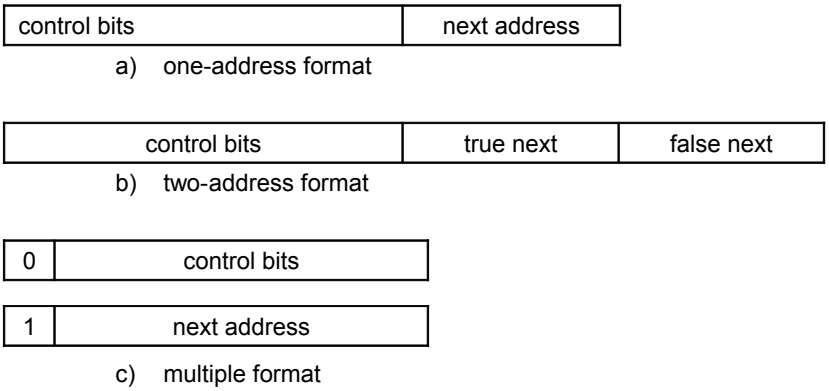


Figure 5.4 Several formats of microword

Horizontal microprogram allows each control bit to be independent from other therefore enables maximum simultaneous events and also offers great flexibility. It is also waste a lot bit. For each field of a microword, there may be a group of

bits that are not active at the same time therefore they can be *encoded* to use a fewer bit. A decoder is required to *decode* these bits and to connect them to the data path. This approach is called *vertical microprogram*. There are many possibilities to compact the micro memory to be as small as possible, sometimes trading off speed for space, for example, a *two-level microprogram*. The first level is *vertical* i.e. maximally encoded; the microword of the level one is pointed to the *horizontal word* of the second level. This is rather like the first level is composed entirely from *subroutine call* and the second level is the subroutine.

The control unit just described has the output of control unit directly mapped to control signals. It is possible to have more than one output format that map the output of the control unit to the actual control signals, for example using the first bit to choose the format, 1 to mean the control bit, 0 to mean the test and jump to other microprogram address. This is called *two-format microprogram*. It is the effort to reduce the size of microprogram because control and test can be mutually exclusive. Other variation is *nano-program* where the group of microprogram word is regarded as a reusable subroutine then the "program" becomes the code to call these subroutines. You can read about the historical record of microprogram era of architecture in [SIE82].

5.3Realisation of microprogrammed systems

This section discusses the equivalence of hardware and software in realising a sequential system. This concept will be illustrated by a simple example of designing a 4-bit comparator in both hardwired and microprogrammed systems (this example is due to [MAN92]).

An assembly of logic elements, whether combinational (AND, OR, NOT, NAND gates, demultiplexors, multiplexor etc) or sequential (flip-flops, registers etc.) is called a *hardwired logic*. By incorporating memories and the content of memory in the test or assignment elements, the system is called a *microprogrammed logic system*; the content is the microprogram. A microprogrammed system can be used to realise a synchronous sequential system, that is, it can be used to implement a control unit.

Example A 4-bit comparator input : A0, A1, B0, B1, and the output Z is the value { EQ, LT, GT }. The logic expression of Z can be described as follows.

Z = (A1' B1' A0 B0' + A1 B1' + A1 B1 A0 B0') . GT + (A1' B1' A0' B0' +
A1 'B1' A0 B0 + A1 B1 A0' B0' + A1 B1 A0 B0) . EQ + (A1' B1' A0'
B0' + A1' B1 + A1 B1 A0' B0) . LT

Where A' is NOT A
The expression can be tabulated in the table below.

number	A1	B1	A0	B0	Z
0	0	0	0	0	EQ
1	0	0	0	1	LT
2	0	0	1	0	GT
3	0	0	1	1	EQ
4..7	0	1	X	X	LT
8..11	1	0	X	X	GT
12	1	1	0	0	EQ
13	1	1	0	1	LT
14	1	1	1	0	GT
15	1	1	1	1	EQ

This expression can be represented as a diagram of *test* and *assignment* primitives that is traversed sequentially by using synchronous sequential system which each clock reads an element of the diagram and executes the primitive (Fig. 5.5).

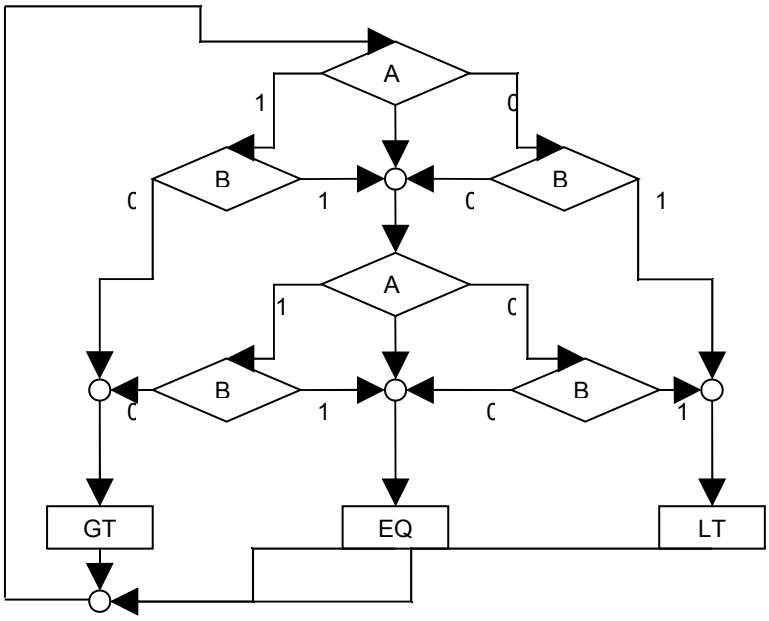


Figure 5.5 The diagram of Z = compare(A,B)

Each primitive (*test*, *assignment*) can be described as follows.

test

if V is true then goto ads1 else goto ads0 (Fig. 5.6).

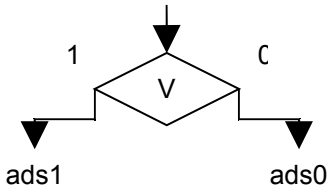


Figure 5.6 The test element

assignment

output OUT and goto next (Fig. 5.7).

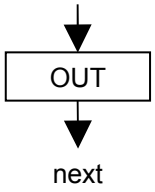


Figure 5.7 The assignment element

The diagram in Fig. 5.5 can be translated into a microprogram as follows (*R* is the output).

```
0     if A1 then goto 1 else goto 2
1     if B1 then goto 3 else goto 6
2     if B1 then goto 8 else goto 3
4     if A0 then goto 4 else goto 5
5     if B0 then goto 7 else goto 6
6     R = GT goto 0
7     R = EQ goto 0
8     R = LT goto 0
```

Next, the microprogram is encoded to map the primitives to a concrete representation. The four cases of test inputs {A1 B1 A0 B0} are encoded into 2 bits (i1, i0). The output { EQ LT GT} is encoded into 3 bits (z2, z1, z0) using unary code.

input	i1	i0
A1	0	0
B1	0	1
A0	1	0
B0	1	1

output	z2	z1	z0
GT	1	0	0
EQ	0	1	0
LT	0	0	1

The microword has two types: *test*, *assignment*. The address field has 4 bits to cover the whole microprogram address (0 . . 8). Fig. 5.8 shows the format of a test microword and an assignment microword.

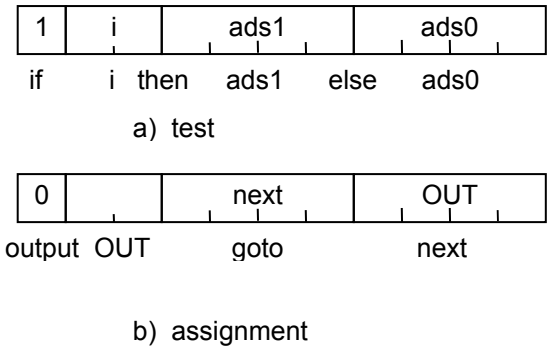


Figure 5.8 The microword format for compare

The microprogram then can be written as follows, where T is the bit specifying the type of a microword.

ads	T	i1 i0	ads1, next	ads0, z
0000	1	0 0	0001	0010
0001	1	0 1	0011	0110
0010	1	0 1	1000	0011
0011	1	1 0	0100	0101
0100	1	1 1	0111	0110
0101	1	1 1	1000	0111
0110	0	- -	0000	-100
0111	0	- -	0000	-010
1000	0	- -	0000	-001

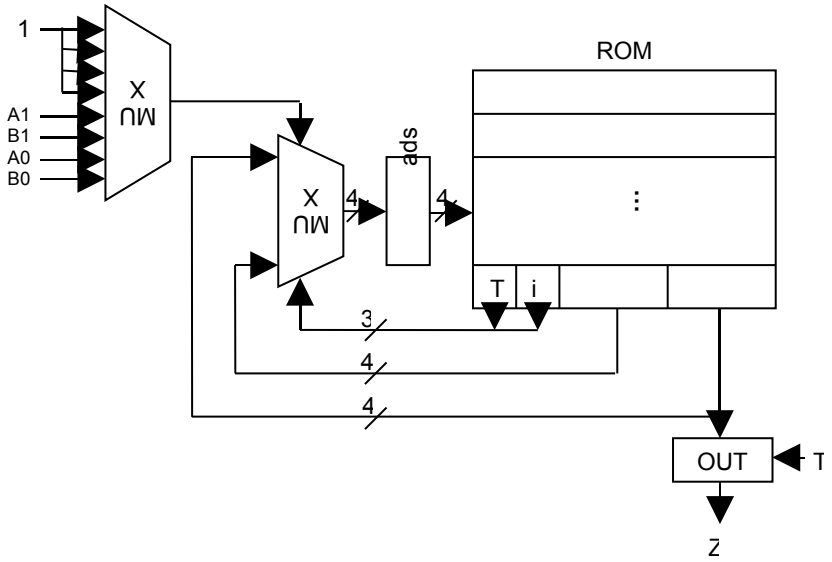


Figure 5.9 The microprogrammed unit to realise the function compare

The microprogrammed unit to realise the function compare is shown in Fig. 5.9. How many cycles it takes to evaluate compare(A, B)? Observing the diagram in Fig. 5.5, on the longest path, there are 5 steps to traverse the diagram hence it takes 5 cycles to evaluate this function using the microprogrammed unit above.

5.4Equivalence of hardware and software

The definition of microprogramming is due to Wilkes, who in 1953 suggested a method for designing the control unit of a processor, based on the use of sequence of microwords – a microprogram – held in a read only memory (ROM). In this context, microprogramming is generally understood as the technique of producing interpreters for high-level language.

At that time random access memory (RAM) that was available was much slower than the processor, leads to CISC (Complex Instruction Set Computer) to achieve high speed the microprogram of CISC are organised horizontally; the need to control a complex processing unit requires each microword to consist of a large number of bits, often over 100.

Firmware, specification of a microprogram, is not an interpretation algorithm but a logic system. The concept of vertically organised microprogram follows that each microword is of fewer bits than in horizontally organised microprogram. The resulting simplicity enables a true optimization of the software to be achieved. Firmware is the transformation and equivalence between hardware (logic systems) and software (microprogram). This hardware-software equivalence is a particular case of the equivalence between space and time.

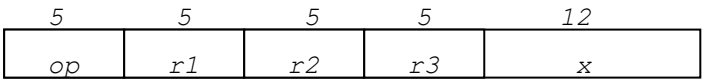
5.5Microprogram for a simple data path

Next we discuss an example of writing a microprogram to control a simple data path. This data path is not a fully functioned processor. However, most of the essential components are present. The example of the control will be the action of fetching an instruction from the memory and executing it.

Data path specification

Data path has a 32-bit ALU. The data width is also 32-bit. It has 32 registers. The register bank has one write port, two read ports. ALU has one function, `add`. The program counter (PC) is 32-bit. It has an increment operation to increment PC. The memory interface unit has two registers: MAR (memory address register), and MBR (memory buffer register). The instruction register (IR) stores the most recent instruction (Fig. 5.10).

The memory contains a code segment storing a program (in machine code). Each instruction is a fixed length 32-bit wide and has the following format, *op* 5-bit, *r1* 5-bit, *r2* 5-bit, *r3* 5-bit, don't care 12-bit.



The field "*op*" specifies operation of the instruction. It has only one operation "*add*" with the code *00001*.

```
ADD  00001
```

r1, r2, r3 specifies a register number, 32 registers requires 5 bits

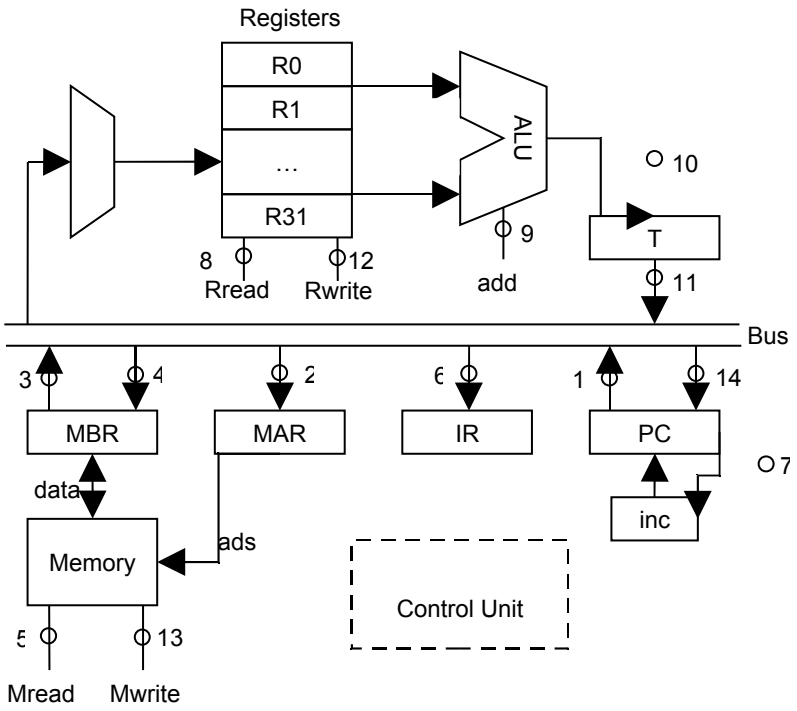


Figure 5.10 A simple data path

Example The instruction "*add*" is written and is encoded into binary bits as follows.

```
add r1 r2 r3
```

means *R[r2]+R[r3] -> R[r1]*

```
00001 00001 00010 00011 x...x
```

The behaviour of the data path can be described in a register transfer level (RTL). The register transfer level specifies how data is flowed between components in the data path. Let's call this description, *microsteps*.

notation

```
<label>
source->destination
```

microsteps

```
<fetch>
 PC -> MAR
 Mread -> MBR
 MBR -> IR
 PC + 1
 decode (goto add)

<add>
 R[r2] + R[r3] -> T
 T -> R[r1]
 goto fetch
```

Each step takes one cycle. Some step can be overlapped if they are independent, for example, PC + 1 can be activated concurrently of any step in *<fetch>*.

Now we want to explore in more details how to realise this behaviour (the microstep). Each connection has an ON/OFF gate (valve) associate with it. The data flow can occur when the gate is ON. To transfer data from a source through bus to a destination, two gates are opened, one is the gate from source to bus, another one is the gate from bus to destination, for example, to do

```
PC -> MAR
```

Two gates are: *PC>BUS* and *BUS>MAR*. The first gate transfers data from *PC* to *BUS*. The second gate transfers data from *BUS* to *MAR*. Here is the list of all gates in the example.

notation *gate number, gate name*

```
1 PC>BUS
2 BUS>MAR
3 MBR>BUS
4 BUS>MBR
5 Mread
6 BUS>IR
7 PC+1
8 Rread
9 ALU:add
10 ALU>T
11 T>BUS
12 Rwrite
13 Mwrite
14 BUS>PC
```

Each microstep can be written as the state of these gates. We use the convention that when a gate's name is written it is activated (or ON) otherwise it is idles (or OFF).

```
<fetch>
 PC>BUS, BUS>MAR
 Mread
 MBR>BUS, BUS>IR
 PC+1

<add>
 Rread, ALU:add, ALU>T
 T>BUS, Rwrite
```

These events can be written using gate numbers as follows.

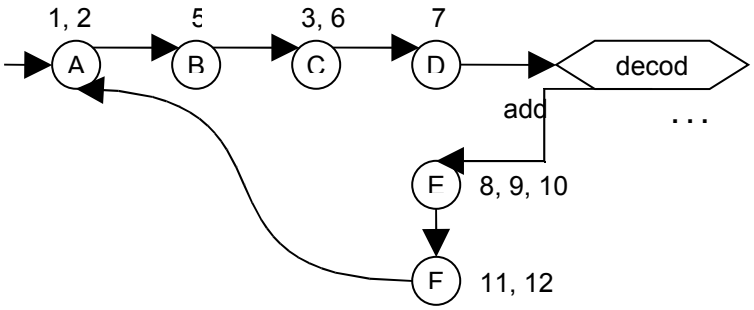
```
<fetch>
 1,2
 5
 3,6
 7
<add>
 8,9,10
 11,12
```

A finite state machine is used to realise the control unit. A FSM for this control unit is shown below. The "*decode*" is a combinational circuit to do a multiway branch to an appropriate execution state according to the opcode-field of an instruction in IR (in this case only one instruction "*add*").

notation: *state {activation; next state}*

```
A {1,2; B}
B {5; C}
C {3,6; D}
D {7; decode}

E {8,9,10; F}
F {11,12; A}
```



This FSM can be implemented using various technologies. The design and implementation of a FSM is greatly simplified by the use of CAD tools.

5.6How complicate is a control unit?

The bound of complexity of a control unit is

$$States \times Control\ inputs \times Control\ outputs.$$

A control unit is implemented as a sequential synchronous machine. It can be described as

$$O = f(I)$$

O output is a function of *I* input, *f* is a Boolean function which is purely combinational. To have state feedback, a part of output is stored in a memory to be fed back to input. This memory is synchronised with a clock so that changes at its output (the state) happen at the edge of clock. Between edges of a clock, its output does not change.

$$Oc = f(Ic,Is)$$
$$Is(t+1) = Os(t)$$

Where *Oc* is the output, *Os* is the state output, *Ic* is the input, *Is* is the state input, *t* is time.

The size of a combination circuit with *I* input and *O* output is $O\,2^{I}$. This is the size required to store the output as a function of inputs.

The control unit in the example has 15 outputs, 5 inputs from opcode-field of IR and 6 states (fetch+add). *Os* must be 3 bits to contain 6 states.

Oc is 15
Ic is 5
Is is 3

So its size is $(15+3)2^{(5+3)}$, approximately 5000 bits (4608). The table that contains *f* and the state feedback can be viewed as a kind of program.

O = f(I)

```
address   O = f(I)   state
I       00110101.... 001
...     10001010.... 010
...     ...
```

The height of this table is 2^{I}, 256 in our example. The width of each row in this table is 15+3 bits. The content of this table is regarded as a program. We can write it down as the event of activation of gates

```
fetch {1,2; B}
B {5; C}
C {3,6; D}
D {7; decode}
add {8,9,10; F}
F {11,12; fetch}
```

This is the microprogram, voila!

5.7Advantage and disadvantage of microprogram

Advantage

Making change to a hardwired control unit implies global change, that is, the circuit will be almost totally changed. Hence, it is costly and time consuming although the present CAD tools do reduce most of the burden in this area. In contrary, for a microprogrammed control unit, making change to it is just changing the microprogram, the bit pattern in the micromemory. There are tools to generate these bits from a human-readable microprogram, hence making change to a microprogram is similar to edit-compile a program. The circuit for control unit does not change. This enables adding new instructions, modifies addressing mode, or updating the version of control behaviour easy to do.

Disadvantage

Microprogram relies on a fast micromemory. It requires a high speed memory. In fact, the architect of an early microprogrammed machine, IBM S360 family, depended on this crucial technology, which was still in the development at that time. The breakthrough in memory technology came, and S360 became the most successful family of computers. A hardwired control unit is much faster. Microprogramming is inherently very low level, making it hard to be absolutely correct. Microprogramming is by nature concurrent, many events occur at the same time, so it is difficult to develop and debug (for a good reading that told a story related to this process, read Tracy Kidder's "The soul of a new machine" [KID00]).

5.8Summary

Microprogram describes the step-by-step execution of the control unit. For our study, this allows the behaviour of the control unit to be changed without changing the circuits. The operational meaning of each instruction can be described using microprogram. By considering steps of control as a program, it creates a new level of abstraction which simplifies the implementation of a complex instruction set. Using this abstraction, building a control unit is similar to writing a program describing the behaviour of an instruction set. We studied various forms of microprogram and their realisation. A systematic method to

realise a general microprogrammed system is illustrated. An example of writing a microprogram for a simple data path is demonstrated.

The technology of microprogram has a big impact on building and designing computers during the era of 1970-1980. Many computer manufacturers adopted this technology. The modern computer aided design tools and the development of modern computer architecture have replaced it. However, microprogramming is an excellent tool to understand computer system behaviour. We will use it to study the processor in the next chapter.

5.9Further Reading

Maurice Wilkes is the inventor of microprogram. His idea was too far ahead of its time as it required a high speed memory which was not possible at that time. Microprogram approach for control unit has several advantages:

1. One computer model can be microprogrammed to *emulate* other model.
2. One instruction set can be used throughout different models of hardware.
3. One hardware can realised many instruction sets. Therefore it is possible to choose the set that is most suitable for an application.

Microprogram becomes obsolete mainly because the present design emphasises the performance and microprogram is slower than hardwired. The change in instruction set design toward a minimum number of cycle per instruction simplifies the instruction set to the point that microprogram is not really required. Also the design of hardwired control unit can be mostly automated as opposed to microprogram which must be written and debug. Hence, for the current instruction set architecture, a hardwired control unit offers a lower engineering cost.

As the history tells us, the design of microprocessors followed the same trend as the earlier computer design. Because of the resource limit (the number of transistor in a chip), hardwired control was implemented and the instruction set architecture was toward a simple design. Ease of change popularised microprogramming. Microprogram made it possible to achieve more complex instruction set. With a much larger micro memory a machine as elaborate as the VAX is possible [LEV89]. In 1984, DEC wanted to offer a cheaper machine with the same instruction set as VAX. They reduced the instructions interpreted by microcode by trapping some instructions and performing them in software. They discovered that 20% of VAX instructions occupied 60% of the microcode,

and yet they are used (executed) only 0.2% of the time. Their simpler subset of VAX, called MicroVAX-1, implemented 80% of VAX instruction in microcode, other 20% is trapped to software, has the size of micromemory reduced from 480K (VAX) to 64K, and perform 90% of the performance of VAX-11/780. This is also an evidence toward a new thinking in instruction set design.

References

[FLY71] Flynn, M., and Rosin, R., "Microprogramming: An introduction and a viewpoint", IEEE Trans. on Computers, July 1971.

[KAT93] Katz, R., Contemporary Logic Design, Addison-Wesley, 1993.

[KID00] Kidder, T., The soul of a new machine, Back bay books, 2000.

[LEV89] Levy, H. and Eckhouse, R., Computer programming and architecture: The VAX, 2nd ed., Digital press, 1989.

[MAN92] Mange, D., Microprogrammed systems: an introduction to firmware theory, Chapman & Hall, 1992.

[WIL53] Wilkes, M., and Stringer, J., "Microprogramming and the design of the control circuits in an electronic digital computer", Proc. of the Cambridge philosophical society, April 1953. Reprinted in [SIE82].

[WIL85] Wilkes, M., Memoirs of a computer pioneer, MIT Press, 1985.

[SIE82] Siewiorek, D., Bell, C., and Newell, A. Computer structures: Principles and examples. McGraw-Hill, 1982.

Exercises

5.1 Using the simple data path in this chapter, write a microprogram to perform the "swap two registers" operation.

5.2 This is one of my favourite exercises. Suppose we want to write a program to perform a search in a list. We will build a machine specifically to do this task. Assuming that the data structure has been loaded into the memory by some mechanism. Using the simple data path in this chapter, write a microprogram to perform this task WITHOUT the

"program" in the code segment. That is, the entire program is in the control unit, or it is in the form of microprogram only.

5.3 To speed up the execution of microprogram, some designer used a *delay* branch, that is, the goto field of microprogram will be delayed by one cycle during the test condition is activated. Design and demonstrate this mechanism. How it will affect the microprogram?

5.4 There are many variations of microprogram: two-level microprogram, nano-program. They are a *compressed* form of describing step-of-control. Please suggest a design of microprogram system that is aimed to be minimal in terms of the size of the microprogram.

5.5 Modern processors have pipeline. Pipeline allows concurrent execution of functional units in the data path. How the step of execution of such data path is described using microprogram?

5.6 Modern processors have many functional units, that is, they are superscalar machines. The instruction stream is reordered such that these functional units can be used as much as possible at the same time. This is a kind of *packing* many instructions into one long instruction for concurrent execution. Please describe a microprogrammed version of such machine. Is it simpler or more complex?