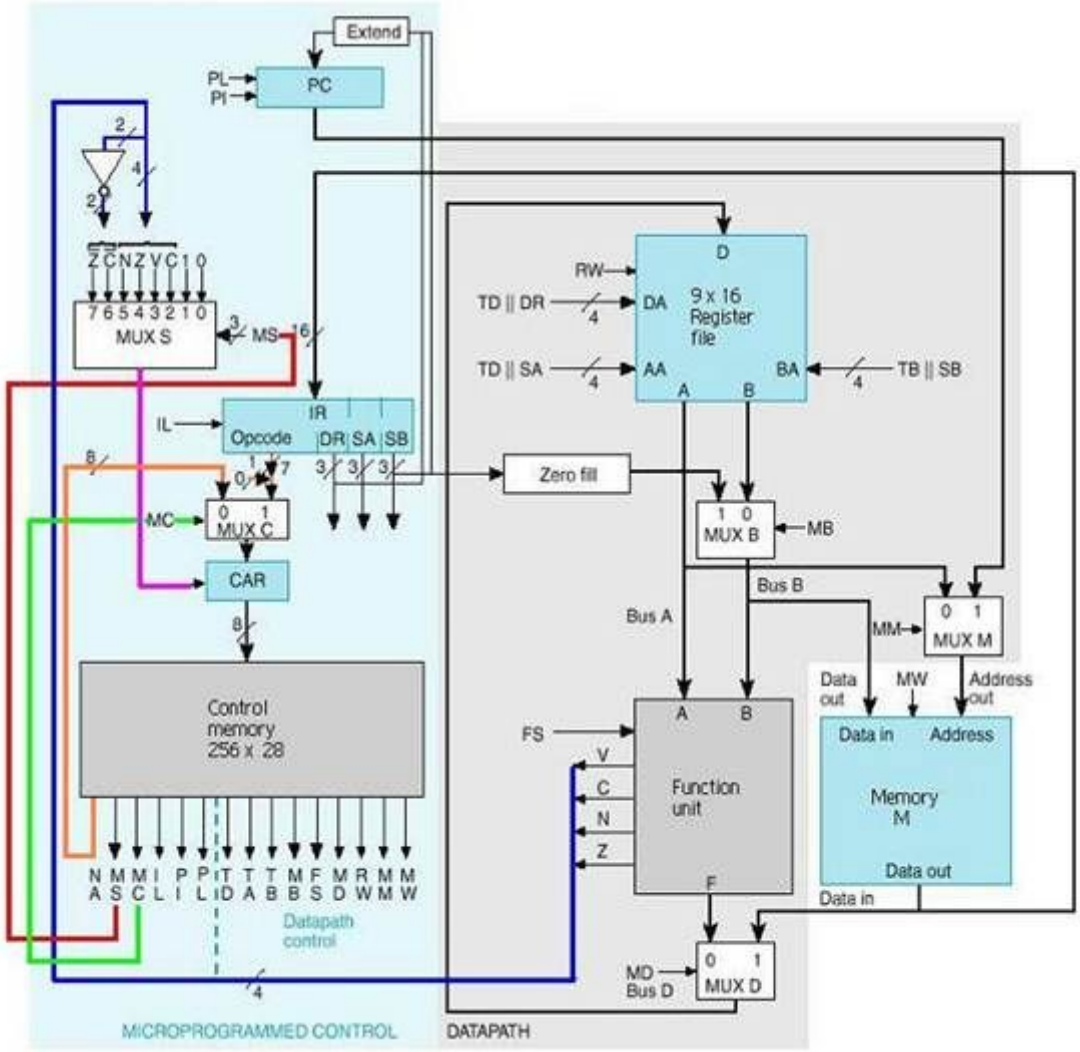
How to Microprogram

Computer Architecture



Main Memory

Main memory contains sequences of instructions, similar to something like an ARM assembly program. For example

```
ADD r0, r1, r2
SUB r3, r0, #4
LDR r0, [r3]
```

could be three consecutive instructions in main memory which form a program. These have to be translated into machine code based on the following:

Opcode	DA	SA	SB
7 bit	3 bit	3 bit	3 bit

For example

`ADD r0, r1, r2 =`

0000011	000	001	010

`= 0000011000001010  = 0x060A`

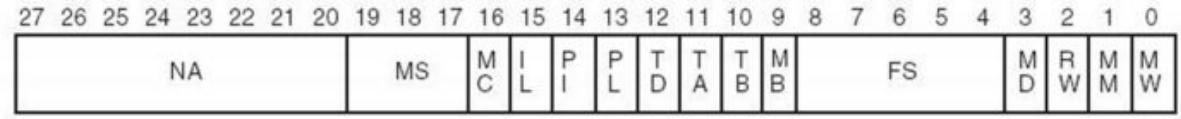
Determining the opcode is discussed below.

Control Memory

Each instruction (eg `ADD`) takes several clock cycles to perform. In fact, fetch, decode, and execute can take multiple clock cycles to perform each!

When an instruction is run, the corresponding **microprogram** is run. Think of this like a subroutine that performs the instruction. Microprograms consist of a sequence of **control words**. Each control word specifies the state of the processor for a single cycle.

Control Word Layout



NA = Address of next control word
MS = Select which status flag to use
MC = Select between NA and IR opcode as CAR input
IL = Load instruction into IR
PI = Increment PC
PL = Offset PC
TD = Use temp reg as dest reg
TA = Use temp reg as reg A
TB = Use temp reg as reg B
MB = Select between immediate value and reg B input B
FS = Select function unit operation
MD = Select between function out and data in for reg file input
RW = Write to dest reg
MM = Select between reg A and PC fo when r memory address input
MW = Write value to selected memory address

Registers

Program Counter (PC)

This contains the address in main memory of the instruction currently being performed. You are probably familiar with the concept of the PC from ARM.

Each clock cycle the value in the PC does one of the following:

1. Unchanged (PL = 0, PI = 0)
2. Incremented (PL = 0, PI = 1)
3. Offset (PL = 1, PI = 0)

It does **not** change every clock cycle, only when moving onto the next instruction in the main memory program.

Instruction Register (IR)

This contains the 16 bit instruction currently being performed.

Each clock cycle the IR does one of the following:

1. Unchanged (IL = 0)
2. Loaded into from memory (IL = 1)

It does **not** change every clock cycle, only when moving o nto the next instruction in the main memory program.

Control Address Register (CAR)

This contains the address in control memory of the control word currently being performed. You can think of it as the PC for microprograms.

Each clock cycle the CAR does one of the following:

1. Incremented (Mux S out = 0)
2. Loaded into from Mux C (Mux S out = 1)

This register **does** change every clock cycle. This is because the control words specify the state of the processor for **one cock** cycle.

Processor Operation

On initialisation of the processor the CAR would be given an initial value. This would be the address in control memory of the first control word in a microprogram.

Fetch Next Instruction microprogram

This microprogram will fetch the next instruction from main memory. This consists of a three control word sequence:

1. Increment the PC and load the instruction at this address from main memory.
2. Write this instruction into the IR.
3. Write the opcode from this instruction into the CAR.

Note that all three are necessary as PC, IR and CAR are each clocked and one feeds into the next sequentially.

Once the opcode of this instruction is in the CAR, it will start performing the microprogram for that instruction on the next clock cycle.

Explicitly in terms of control word values:

NA	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
x	000	x	0	1	0	x	x	x	x	x	x	0	1	0
x	x	x	1	0	0	x	x	x	x	x	x	0	1	0
x	001	1	0	0	0	x	x	x	x	x	x	0	1	0

Where **x**'s are don't cares. Those can just be set to 0 for convenience.

So the fetch microprogram would be stored in control memory starting at address `0x10` as:

Address	Value
0x10	0x0004002
0x11	0x0008002
0x12	0x0030002

Using this microprogram

The first way would be to use it for a main memory instruction. To do this you would make an instruction with it's opcode being the first address in the microprogram. In our case the 7 bit opcode would be 001000 (0x10).

The second way would be to use it within another microprogram. Typically at the end of each microprogram you need to fetch the next instruction (unless doing a jump or branch). It therefore makes sense to call this microprogram at the end of other microprograms. This can be achieved with the following control word:

NA	MS	MC	IL	PI	PL	TD	TA	TB	MB	FS	MD	RW	MM	MW
0001000	001	0	0	0	0	x	x	x	x	x	x	0	1	0

Notice that NA is targeting address 0x10 as in our example above.

For example, you could have a sequence of control words to perform some operation such as multiplication and then follow it with the control word above to run the fetch next instruction microprogram.mux

Branching

If however a branch instruction was run the PC would not be incremented. Instead the current PC would be offset by some value (the value from the ‘Extend’ unit) to jump somewhere else in main memory. In this case PL would be set in place of PI. Otherwise, an unconditional branch instruction and a fetch next instruction instruction would have similar microprograms.

In the case of a conditional branch, the MS select value would depend on the status flag being selected. For example for a branch if equal (BEQ) instruction, MS would be 100 to select the Z flag. The value of the Z flag would then decide whether the CAR just increments (if Z = 0) or if it loads the unconditional branch microprogram (if Z = 1). If the CAR does just increment, the following control word would be the fetch next instruction microprogram.

i.e the two control word sequence for conditional branch would be:

1. Go to unconditional branch microprogram (if Z = 1)
   or go to next control word (if Z = 0)
2. Go to ‘fetch next instruction’ microprogram