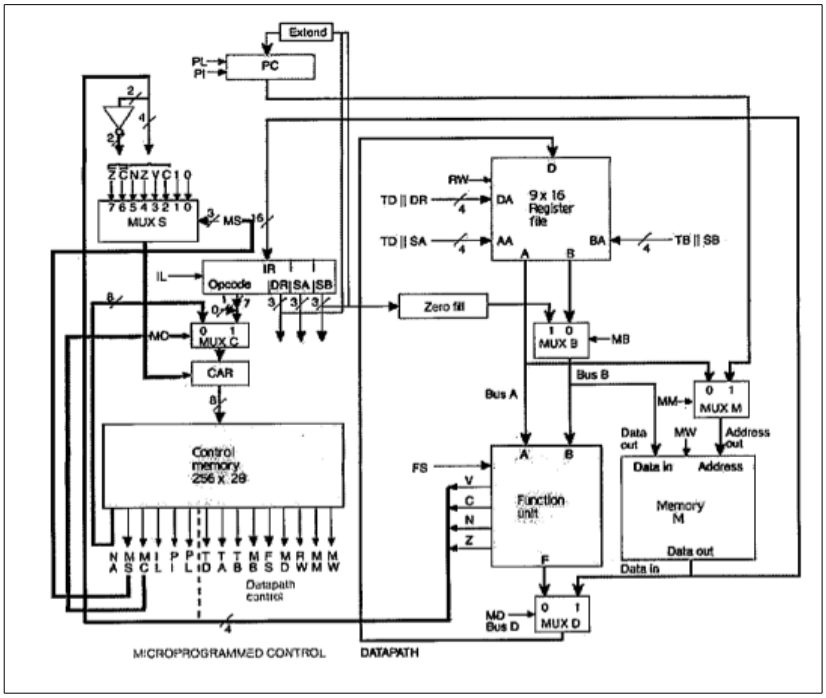
# Computer Architecture – 2016 Solutions

## Question 1

*(1)*

*a) Explain in detail the operations that take place when the following multiple cycle microprogrammed instruction set processor executes machine instructions. Your explanation must include operations in the processor's control e.g how does one instruction in the IR register execute several control words in the Control Memory.*




## Setup

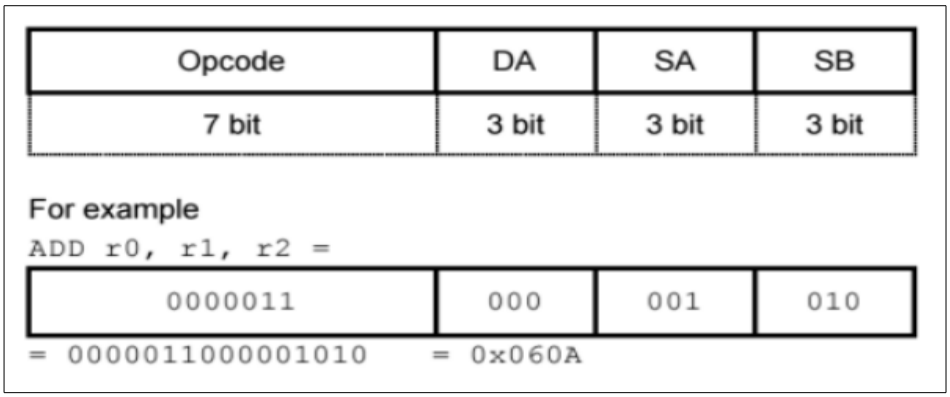
When the system is first turned on the Program Counter (PC) and the Control Access Register (CAR) are **RESET**. This is to ensure all addresses and values within these devices are reset to the desired start off values.

The Program Counter (PC) specifies the address of the next microprogram stored within the Memory Unit (M), while the Control Access Register (CAR) specifies the address of the next set of machine code instructions within the Control Memory. The PI signal increments the value of the PC, and the PL signal uses DR || SB (concatenated) as the next address.

When the PC is **RESET** it is loaded with the address of the first microprogram within the Memory Unit (M). Memory contains the addresses of instructions that are stored within the Control Memory. For example, instruction 0x002E from the Memory Unit (M) may correspond to the Micro Code for an ADI instruction located at a given address within the Control Memory.

**Execution**

The microprogram (instruction) address is loaded from the Memory Unit (M) into the Instruction Register (IR), where it is decoded and split into a 7-bit Opcode that corresponds to the address of the instruction to be accessed within the Control Memory. It is also split into 3 x 3-bit vectors that can either be used for register selection or as values (either immediate or PC offset for branching). They decide the Destination Register (DA).




The Opcode is then loaded into MUXC, and, as the control memory is in the Instruction Fetch mode, it is loaded through MUXC and into the Control Access Register (CAR). Here the Opcode is used to determine the next address the Control Memory uses to access the next instruction.

The Control Memory accesses the instruction to be accessed (as dictated by the CAR) based on the instruction address passed in. An instruction could take more than one clock cycle (e.g LOAD).

The Control Memory controls the input signals to all of the multiplexers, Function Unit, Register File, PC, IR, CAR and Memory.

Values coming from either the output of the Function Unit or Memory (decided by MUXD) can be loaded into registers within the Register File if RW (Read=0/Write=1) is set to high. The Register File contains an array of registers which hold data. A temporary register (R8) is also here for multi-cycle instructions so that the data the user may be storing is not disrupted. The Destinartion Register is selected by DR coming from IR, or TD (for temp register) coming from Control Memory.

The Function Unit takes in values from the Register File (selected by SA, SB, TA or TB) or immediate values (passed from IR through MUXB). It then performs arithmetic, logic or shifting operations on these values. It contains an Arithmetic Logic Unit (ALU) and a Shifter. The status bits from the Function Unit are passed through to MUXS and are used for conditional instructions within the Microprogrammed Control.

Values from the Register File can also be written and stored into the Memory Unit (M) if MW (Memory Write) is set to high.

The processor is pipelined so that a number of operations can be happening simultaneously. The system could simultaneously be fetching the next instruction, executing a previous instruction and writing the results to the Register File.

*b) How would you micro-code a conditional branch instruction?*

If 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 to somewhere else in memory. In this case PL would be set in place of PI.

In the case of a conditional branch, the MS (MUXS with flags) select value would depend on the status flag desired. For example for a branch-if-equal (BEQ) instruction, MS would be 100 to select the Z flag. The value of the Z flag (1 for equals, 0 for not equals) would then decide whether the CAR just increments (if Z=0) or if it loads the unconditional branch instruction (if Z=1). If the CAR just increments, the following control word would be the fetch next instruction microprogram.

The control word sequence for a conditional branch would be:

---

*If (Z=1)*

*Go to unconditional branch microprogram*

*Else If (Z=0)*

*Go to next control word*

---

Following the execution of this conditional block the next executed microprogram would be the *fetch next instruction* microprogram.

**Question 2**

*(2)*

*a) Provide a detailed schematic for a Function Unit that implements the following micro operations:*

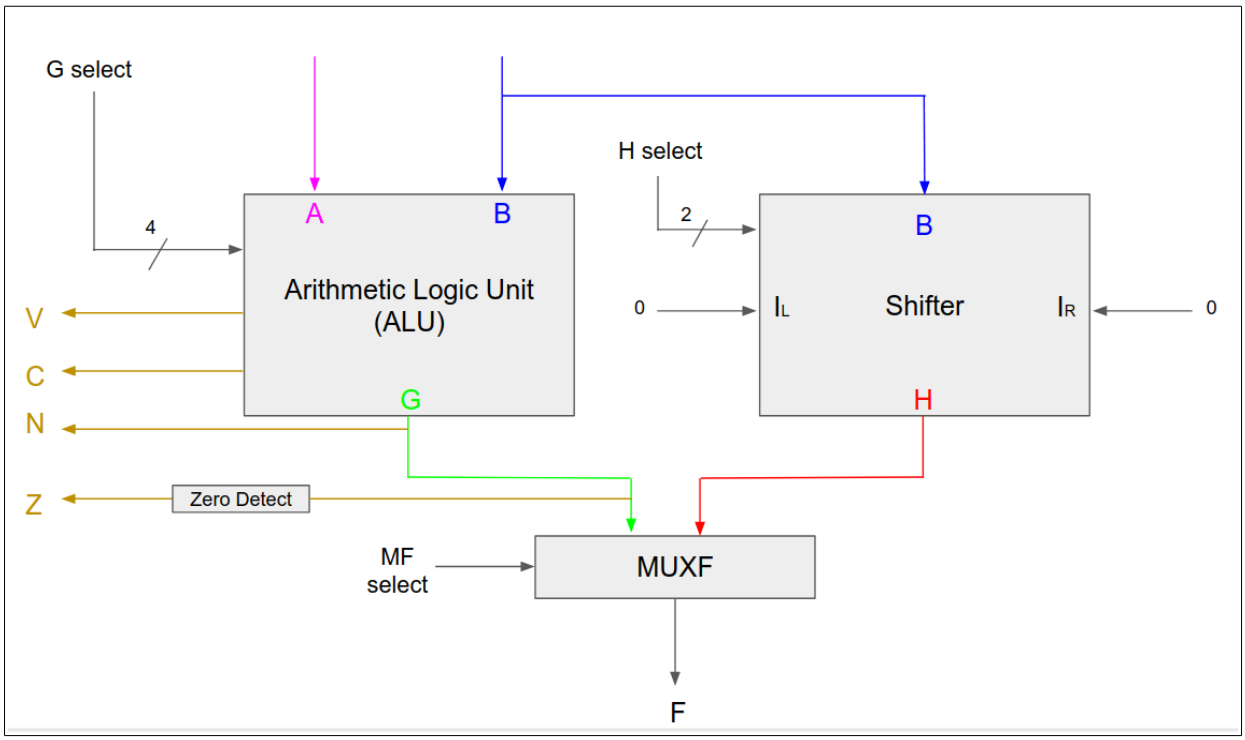
Table 1: FS code definition

| FS | Micro-operation |
|---|---|
| 00000 | $F = A$ |
| 00001 | $F = A + 1$ |
| 00010 | $F = A + B$ |
| 00011 | $F = A + B + 1$ |
| 00100 | $F = A + \bar{B}$ |
| 00101 | $F = A + \bar{B} + 1$ |
| 00110 | $F = A - 1$ |
| 00111 | $F = A$ |
| 01000 | $F = A \wedge B$ |
| 01010 | $F = A \vee B$ |
| 01100 | $F = A \oplus B$ |
| 01110 | $F = \bar{A}$ |
| 10000 | $F = B$ |
| 10100 | $F = srB$ |
| 11000 | $F = slB$ |

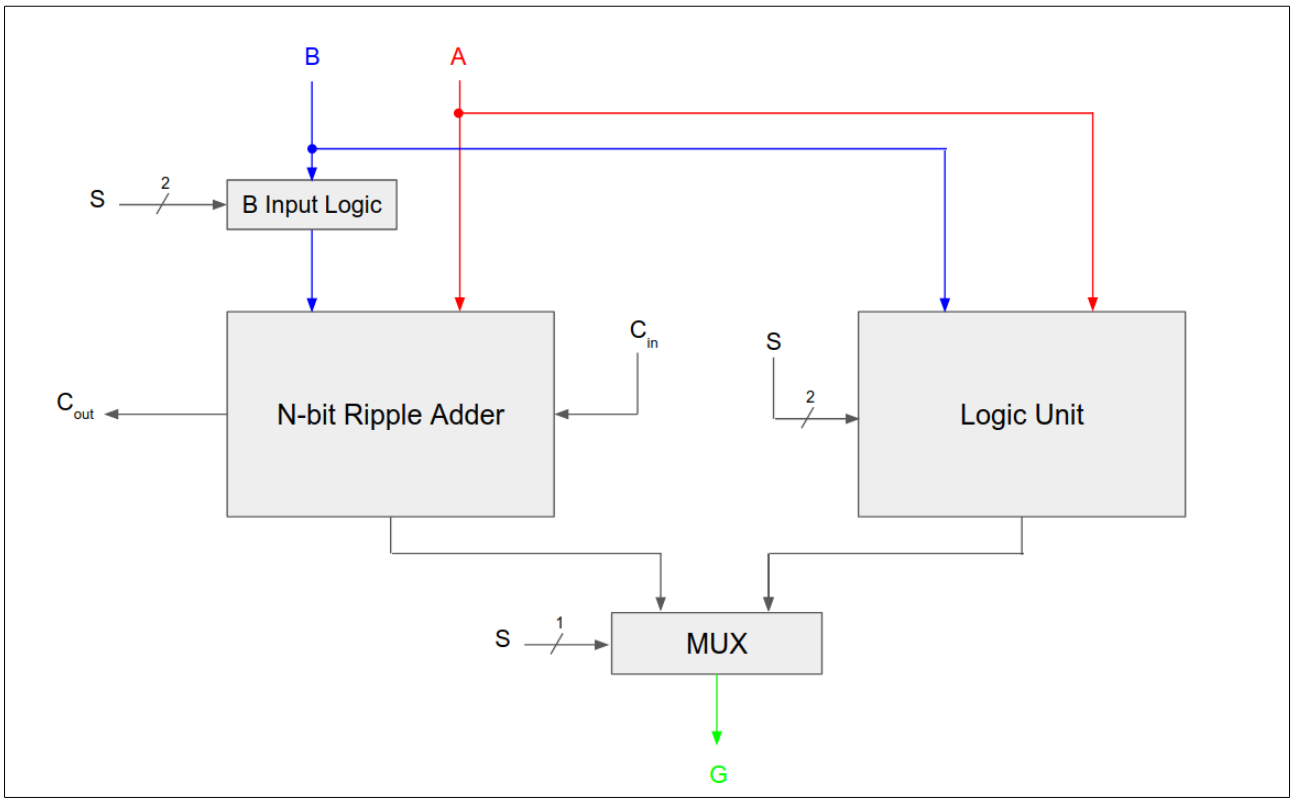
A Function Unit consists of the following components:

- 16bit Arithmetic Logic Unit (ALU):
  - N-Bit Ripple Adder
    - N x Full Adders
  - Logic Unit
  - B Input Logic
- 16bit Shifter:
- 2 to 1 line multiplexer:

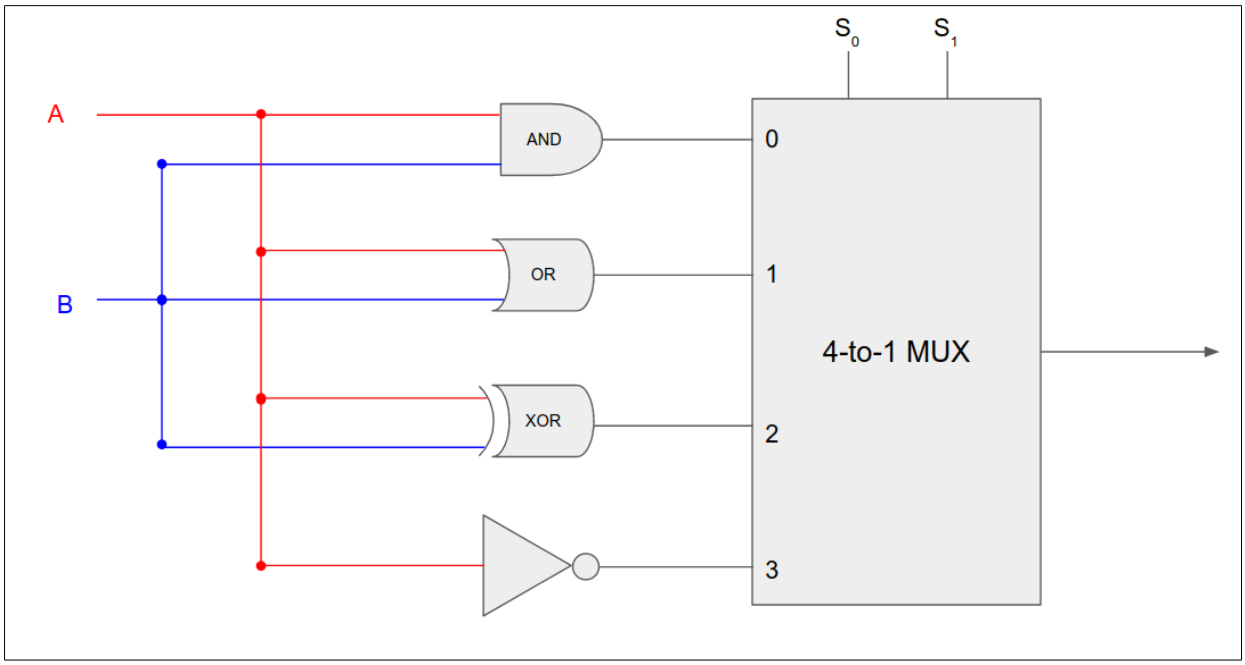
1. *Function Unit Overview*



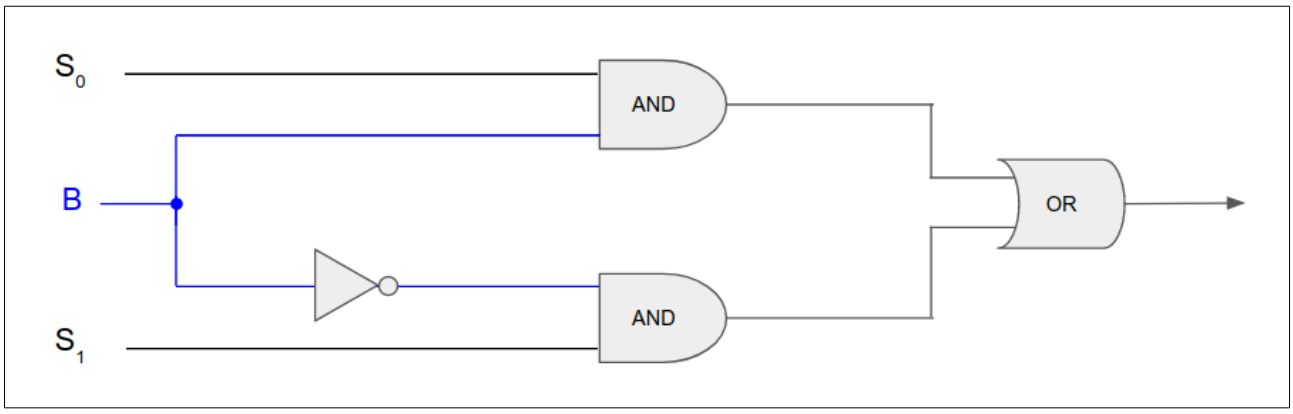
## 2. 16-Bit Arithmetic Logic Unit (ALU)



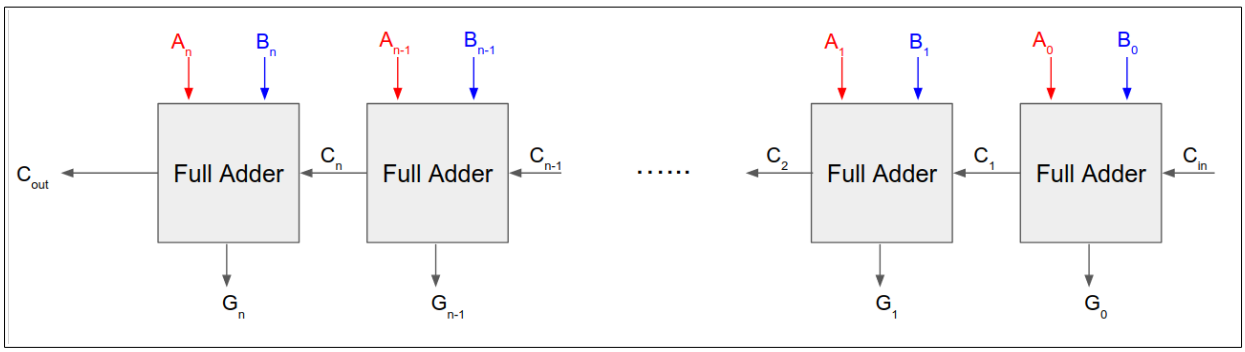

## 3. Logic Unit



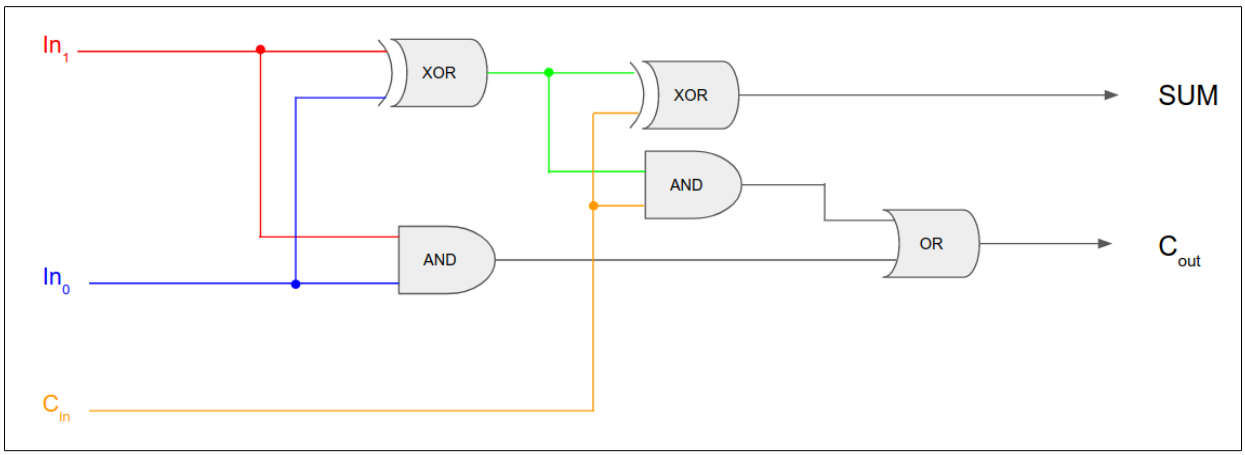
## 4. B Input Logic (1-bit Slice)



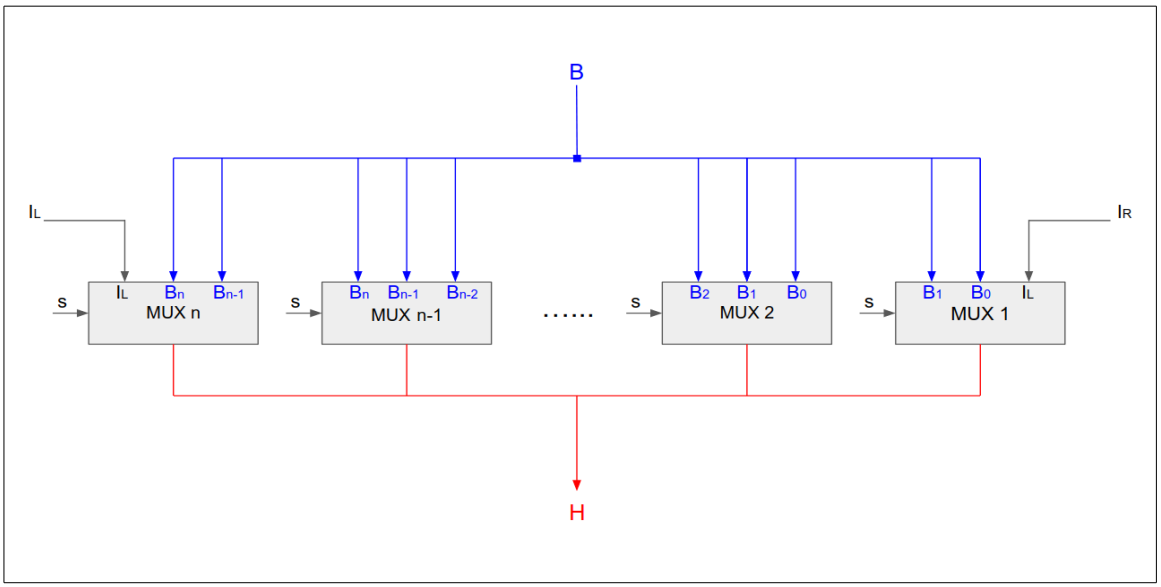

## 5. Ripple Adder



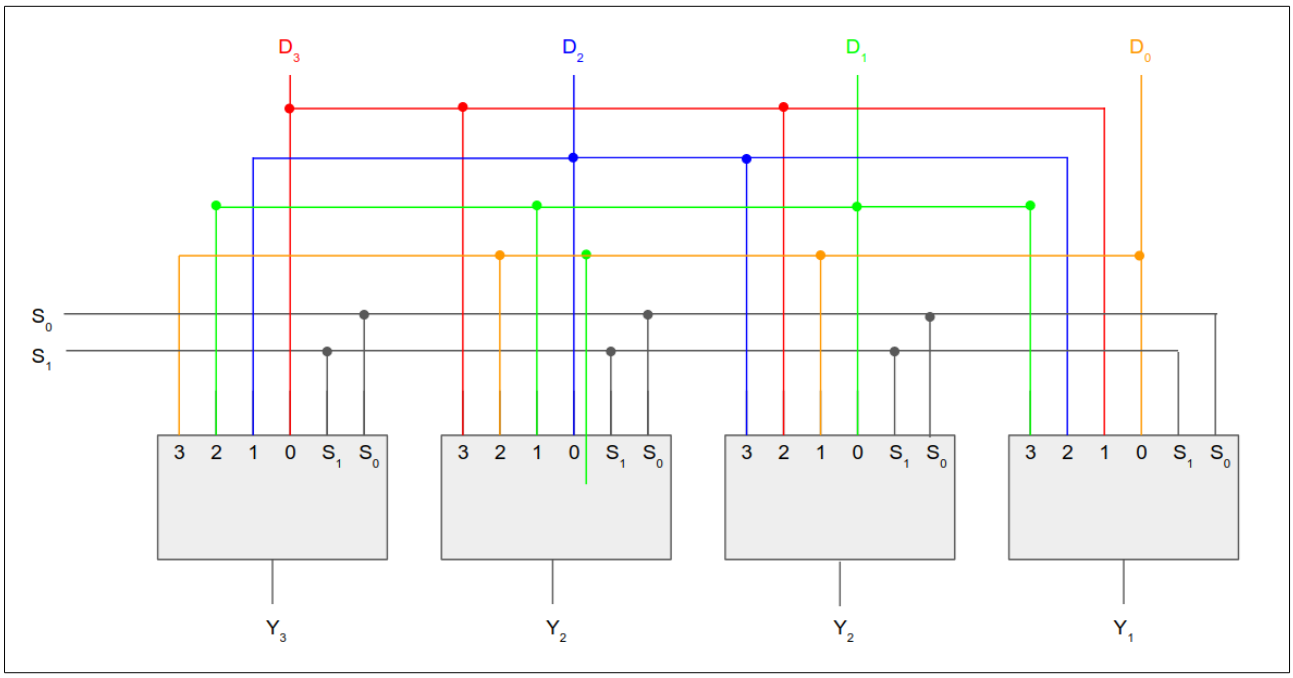

## 6. Ripple Adder



## 7. 16-Bit Shifter Schematic




*b) How would you implement a barrel shifter? Please provide a schematic for a 4 bit barrel shifter.*



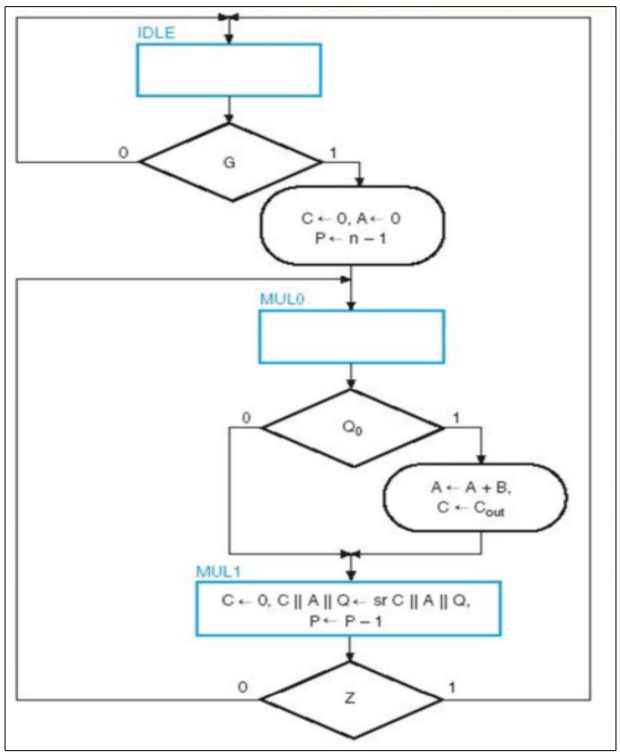

Barrel shifters are able to perform multiple shifts. It is implemented by supplying select signals to a 4 x 4-1 Multiplexers. Based on the values of $S_0$ and $S_1$ the output data Y will rotate and wrap around. The table below shows how multiple shifts may be performed:

| $S_0$ | $S_1$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ | Micro-Ops |
|---|---|---|---|---|---|---|
| 0 | 0 | $D_3$ | $D_2$ | $D_1$ | $D_0$ | No Rotate |
| 0 | 1 | $D_2$ | $D_1$ | $D_0$ | $D_3$ | Rotate One |
| 1 | 0 | $D_1$ | $D_0$ | $D_3$ | $D_2$ | Rotate Two |
| 1 | 1 | $D_0$ | $D_3$ | $D_2$ | $D_1$ | Rotate Three |

**Question 3**

*(3)*

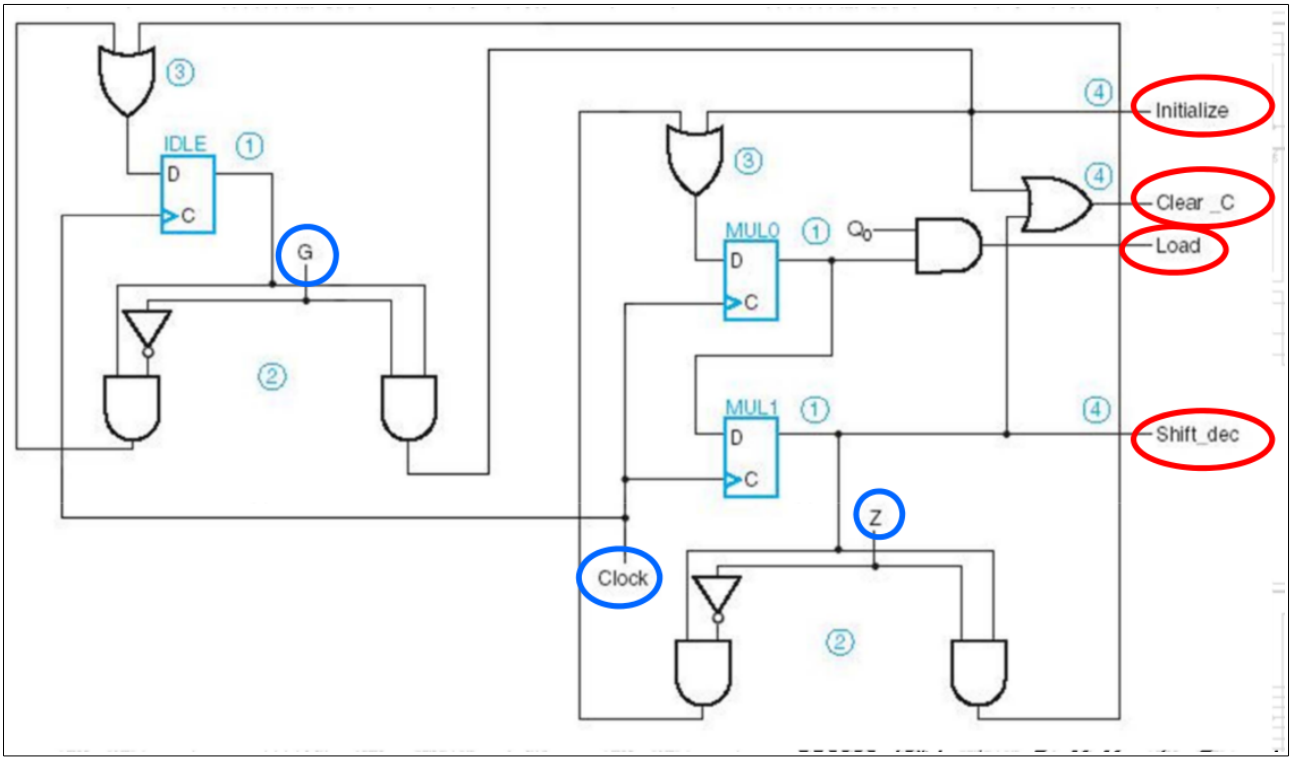
*a) Provide a schematic for the Binary Multiplier data path with a One Flip-Flop per State control unit.*



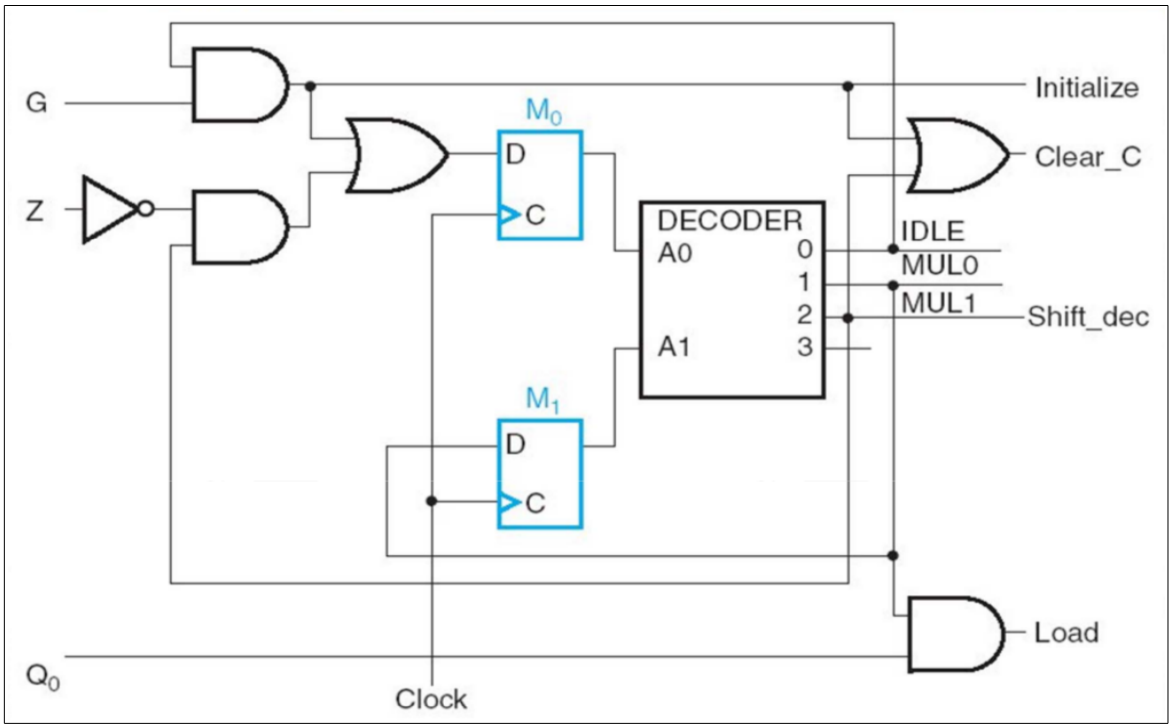

**Converting from ASM to Flip-Flop Circuit:**

- Junctions → OR Gates
- Decision Boxes → Demultiplexers
- State Boxes → Flip-Flops

**Flip-Flop Circuit:**



*b) Provide a schematic for the Binary Multiplier data path with a Sequence Register and Decoder control unit.*




- Within the Control Unit, implement two D-Flip flops and a 2 to 4 decoder. These will be used to control the State signals as can be seen on output.

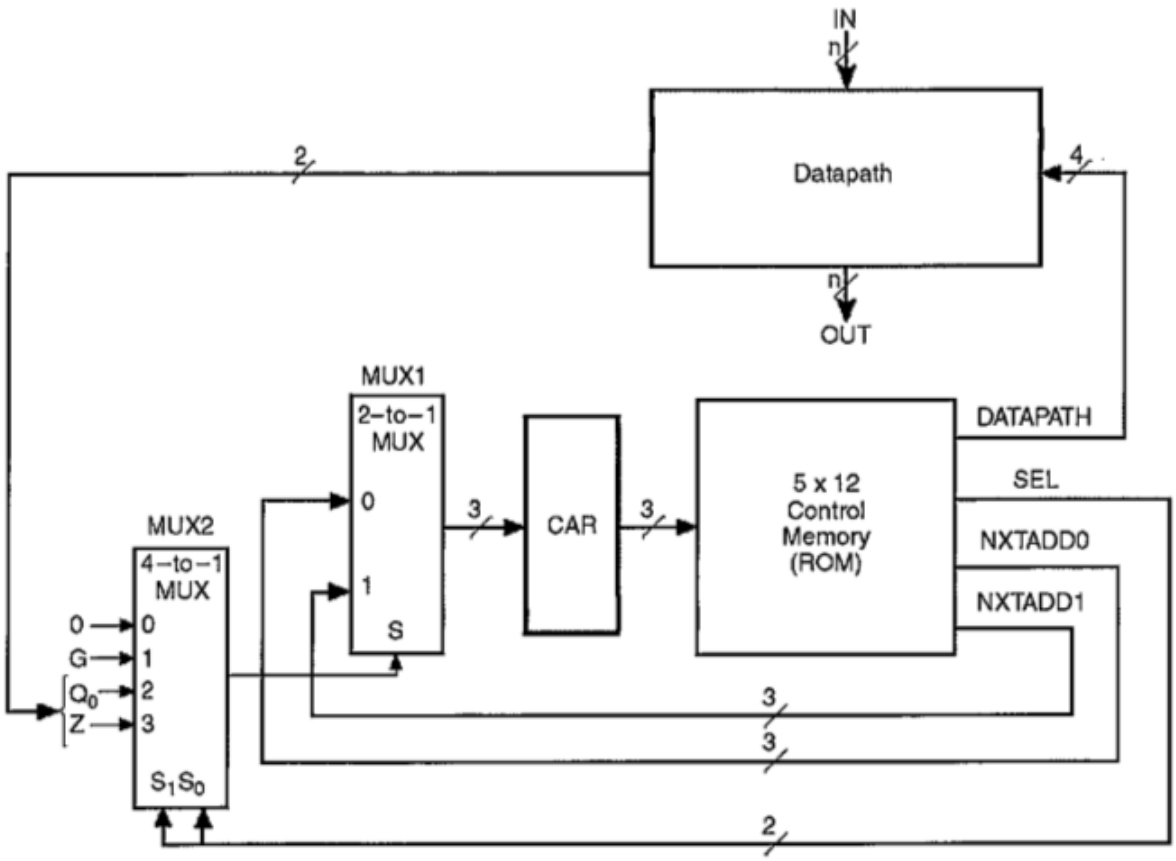
- $D_{M0}$ = (IDLE & G) or (MUL1 & !Z)
- $D_{M1}$ = MUL0

| Present state | | | Inputs | | Next state | | Decoder Outputs | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | $M_1$ | $M_0$ | G | Z | $M_1$ | $M_0$ | IDLE | MUL0 | MUL1 |
| IDLE | 0 | 0 | 0 | × | 0 | 0 | 1 | 0 | 0 |
| | 0 | 0 | 1 | × | 0 | 1 | 1 | 0 | 0 |
| MUL0 | 0 | 1 | × | × | 1 | 0 | 0 | 1 | 0 |
| MUL1 | 1 | 0 | × | 0 | 0 | 1 | 0 | 0 | 1 |
| | 1 | 0 | × | 1 | 0 | 0 | 0 | 0 | 1 |
| — | 1 | 1 | × | × | × | × | × | × | × |

**Question 4**

*(4)*

*a) Provide Micro-Code for the following Microprogrammed Control Unit. This unit controls a Binary Multiplier.*




| Control Signal | Register Transfers | States in Which Signal is Active | Micro-instruction Bit Position | Symbolic Notation |
|---|---|---|---|---|
| Initialize | $A \leftarrow 0, P \leftarrow n-1$ | INIT | 0 | IT |
| Load | $A \leftarrow A+B, C \leftarrow C_{out}$ | ADD | 1 | LD |
| Clear_C | $C \leftarrow 0$ | INIT, MUL1 | 2 | CC |
| Shift_dec | $C\|A\|Q \leftarrow \text{sr } C\|A\|Q, P \leftarrow P-1$ | MUL1 | 3 | SD |

- The system must be able to respond to two status bits:

- Z (Zero Detect)
- $Q_0$ (Q-Bit)

- One Control Bit:
- G (Go Signal)

-As a result, two **SEL** bits must be included in the control word

| SEL | | |
|---|---|---|
| **Symbolic notation** | **Binary Code** | **Sequencing Microoperations** |
| NXT | 00 | $CAR \leftarrow NXTADD0$ |
| DG | 01 | $\overline{G}$: $CAR \leftarrow NXTADD0$<br>$G$: $CAR \leftarrow NXTADD1$ |
| DQ | 10 | $\overline{Q_0}$: $CAR \leftarrow NXTADD0$<br>$Q_0$: $CAR \leftarrow NXTADD1$ |
| DZ | 11 | $\overline{Z}$: $CAR \leftarrow NXTADD0$<br>$Z$: $CAR \leftarrow NXTADD1$ |

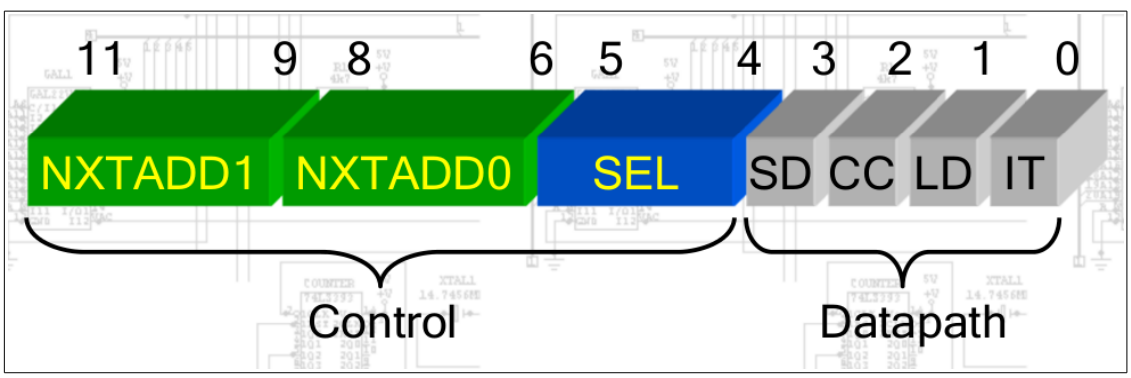
- Finally we add two next-address fields:

- NXTADD0 (Next Address 0)
- NXTADD1 (Next Address 1)

- The control word must also supply four control signals as follows

| Control Signal | Register Transfers | States in Which Signal is Active | Micro-instruction Bit Position | Symbolic Notation |
|---|---|---|---|---|
| Initialize | $A \leftarrow 0, P \leftarrow n-1$ | INIT | 0 | IT |
| Load | $A \leftarrow A+B, C \leftarrow C_{out}$ | ADD | 1 | LD |
| Clear_C | $C \leftarrow 0$ | INIT, MUL1 | 2 | CC |
| Shift_dec | $C \| A \| Q \leftarrow \text{sr } C \| A \| Q, P \leftarrow P-1$ | MUL1 | 3 | SD |

- All of this results in a 12-bit Control Word which could take the following layout:



- Next we design the microprogram in symbolic Register Transfer Form:

| Address | Symbolic transfer statement |
|---|---|
| IDLE | $G$: $CAR \leftarrow$ INIT, $\overline{G}$: $CAR \leftarrow$ IDLE |
| INIT | $C \leftarrow 0, A \leftarrow 0, P \leftarrow n-1, CAR \leftarrow$ MUL0 |
| MUL0 | $Q_0$: $CAR \leftarrow$ ADD, $\overline{Q_0}$: $CAR \leftarrow$ MUL1 |
| ADD | $A \leftarrow A+B, C \leftarrow C_{out}, CAR \leftarrow$ MUL1 |
| MUL1 | $C \leftarrow 0, C \| A \| Q \leftarrow$ sr $C \| A \| Q$, $Z$: $CAR \leftarrow$ IDLE, $\overline{Z}$: $CAR \leftarrow$ MUL0, $P \leftarrow P-1$ |

- This is then translated into micro code:

| Address | NXTADD1 | NXTADD0 | SEL | DATAPATH | Address | NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|---|---|---|---|---|---|
| IDLE | INIT | IDLE | DG | None | 000 | 001 | 000 | 01 | 0000 |
| INIT | — | MUL0 | NXT | IT, CC | 001 | 000 | 010 | 00 | 0101 |
| MUL0 | ADD | MUL1 | DQ | None | 010 | 011 | 100 | 10 | 0000 |
| ADD | — | MUL1 | NXT | LD | 011 | 000 | 100 | 00 | 0010 |
| MUL1 | IDLE | MUL0 | DZ | CC, SD | 100 | 000 | 010 | 11 | 1100 |

---

IDLE – Idle State – [000]

Set the Binary Multiplier to the Idle State.

**Address in Control Memory –** [000]

**Instruction Description -**

- Set NXTADD1 → INIT State
- Set NXTADD0 → IDLE State
- Set SEL → DG (Decison based on G)
- Set Datapath = [ 0 ] [ 0 ] [ 0 ] [ 0 ] *None*

  [SD] [CC] [LD] [IT]

| NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|
| 001 | 000 | 01 | 0000 |

---

INIT – Initialize State – [001]

Initialize the Binary Multiplier.

**Address in Control Memory –** [001]

**Instruction Description -**

- Set NXTADD1 → DON'T CARE
- Set NXTADD0 → MUL0
- Set SEL → NXT (Pass through NXTADD0)
- Set Datapath = [ 0 ] [ 1 ] [ 0 ] [ 1 ] *Initialize & Clear C*
  [SD] [CC] [LD] [IT]

| NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|
| 000 | 010 | 00 | 0101 |

---

MUL0 – First Multiplication Step – [010]

Get system ready to perform multiplication step.

**Address in Control Memory –** [010]

**Instruction Description -**

- Set NXTADD1 → ADD
- Set NXTADD0 → MUL1
- Set SEL → DQ (Decision based on Q)
- Set Datapath = [ 0 ] [ 0 ] [ 0 ] [ 0 ] *None*
  [SD] [CC] [LD] [IT]

| NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|
| 011 | 100 | 10 | 0000 |

---

ADD – Addition Step– [011]

Perform binary multiplication addition step.

**Address in Control Memory –** [011]

**Instruction Description -**

- Set NXTADD1 → DONT CARE
- Set NXTADD0 → MUL1
- Set SEL → NXT (Pass through NXTADD0)
- Set Datapath = [ 0 ] [ 0 ] [ 1 ] [ 0 ] *Load*

  [SD] [CC] [LD] [IT]

| NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|
| 000 | 100 | 10 | 0010 |

---

MUL1 – Multiplication Shift– [011]

Perform binary multiplication shift step.

**Address in Control Memory –** [011]

**Instruction Description -**

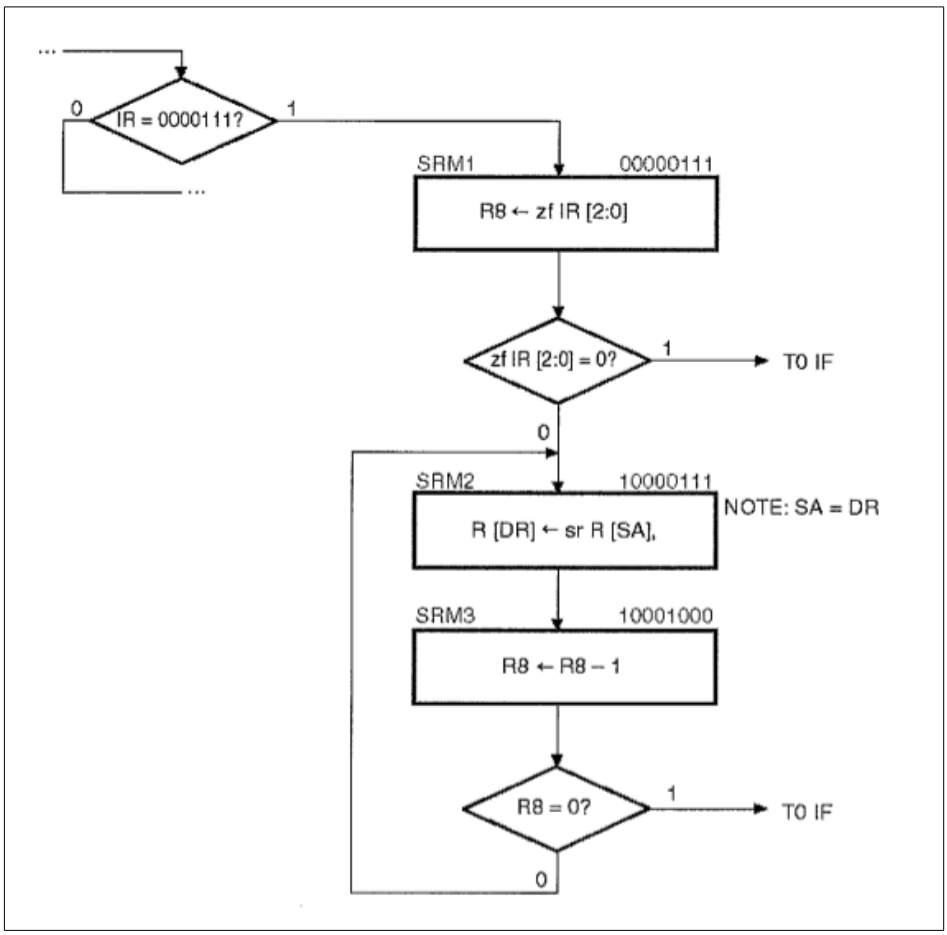
- Set NXTADD1 → IDLE
- Set NXTADD0 → MUL0
- Set SEL → DZ (Decision based on Z)
- Set Datapath = [ 1 ] [ 1 ] [ 0 ] [ 0 ] *Clear C & Shift Dec*

  [SD] [CC] [LD] [IT]

| NXTADD1 | NXTADD0 | SEL | DATAPATH |
|---|---|---|---|
| 000 | 010 | 11 | 1100 |

---

*b) The following ASM defines an instruction for our microprogrammed multi-cycle instruction set processor. What istruction does the ASM define?*




This ASM defines the instruction that occurs when the Opcode 0000111 is passed from the Instruction Register (IR) through to the CAR.

The first instruction accessed is that stored within the Control Memory at address 00000111 named SRM1.

R8 ← zf IR [2:0]

What this defines is an instruction that loads the contents creates a 16-bit value containing 13 0's followed by the contents of IR [2:0]. This value is then stored into Register 8.

The system then performs a comparison checking if the previous value stored in R8 ( zf IR [2:0] ) is equal to 0. If this is true ( zf IR [2:0] = 0), the system branches off to IF ( Instruction Fetch ).

Otherwise, if the previous value stored in R8 does not equal to 0 the system then moves onto the instruction named SRM2 which is stored within the Control Memory at address 10000111.

R [DR] ← sr R [SA]

What this instruction does is it performs an sr (Shift Right) operation on the value stored within R [SA], which is R [DR] according to the note. The result of this shift is then stored within R [DR].

The system then moves onto the instruction named SRM3 which is stored within the Control Memory at address 10001000.

R8 ← R8 -1

This instruction quite simply subtracts 1 from the current contents within Register 8, storing the resultant value once again within R8. The system then checks if R8 = 0. If this is true the system branches to IF (Instruction Fetch). If this is not true, the system loops back around to SRM2 performing the previous shift operations.This is repeated until the value in R8 eventually reaches 0.

What this ASM describes is a shift-right by X operation, which can be translated into ARM assembly code in the following form:

LSR R[DR] , R[DR], #X