

L19-CS8421-Microcode

Instruction Sets & Microcode

CS8421

Computing Systems

Dr. Ken Hoganson

Class

Will

Start

Momentarily...

We need to develop an instruction set and formats that support the following operations:

Mem \leftarrow R1	8 instructions
R1 \leftarrow Mem	8 instructions
R1 \leftarrow R1 OP R2,	16 instructions
R1 \leftarrow R1 OP Mem	32 instructions
R2 \leftarrow R1 OP Mem	32 instructions
Mem \leftarrow R1 OP R2	64 instructions
Mem2 \leftarrow R1 OP MEM1	16 instructions

The machine has 32 general purpose registers, each 16 bits in size. Memory is accessed by a base plus displacement of 10 bits.

<u>Pseudocode</u>	<u>Opcode & Operands</u>	<u>Format</u>	<u>#inst</u>
Mem \leftarrow R1	OP R1 B2 D2	A	8
R1 \leftarrow Mem	OP R1 B2 D2	A	8
R1 \leftarrow R1 OP R2	OP R1 R2	B	16
R1 \leftarrow R1 OP Mem	OP R1 B2 D2	A	32
R2 \leftarrow R1 OP Mem	OP R1 R2 B3 D3	C	32
MEM \leftarrow R1 OP R2	OP R1 R2 B3 D3	C	64
Mem2 \leftarrow R1 OP Mem1	OP R1 B2 D2 B3 D3	D	16

3 or 4 Formats needed, two bits to specify the format.

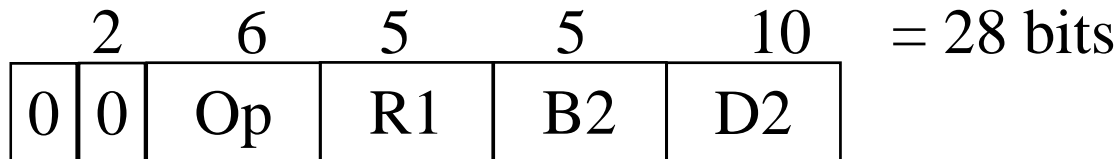
A: 48 instructions

B: 16 instructions

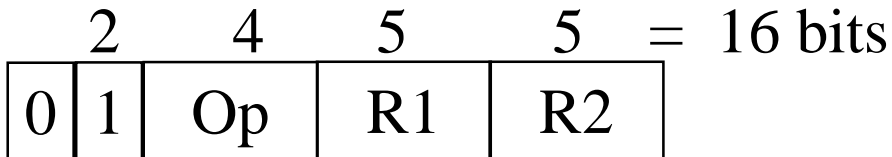
C: 96 instructions

D: 16 instructions

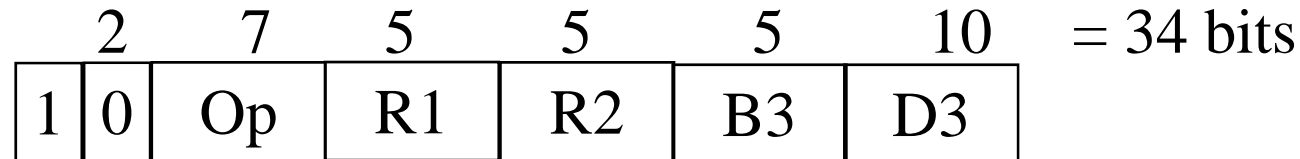
Format A: 48 inst



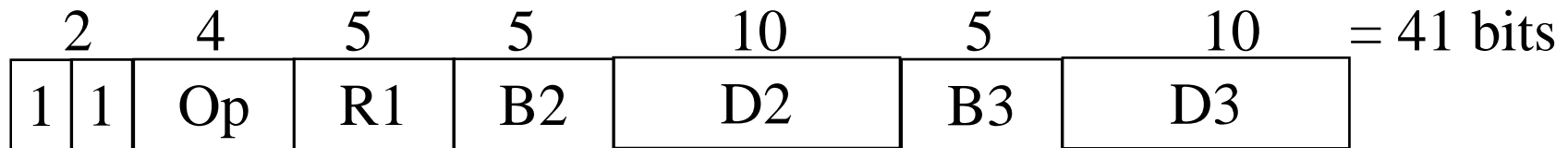
Format B: 16 inst



Format C: 96 inst



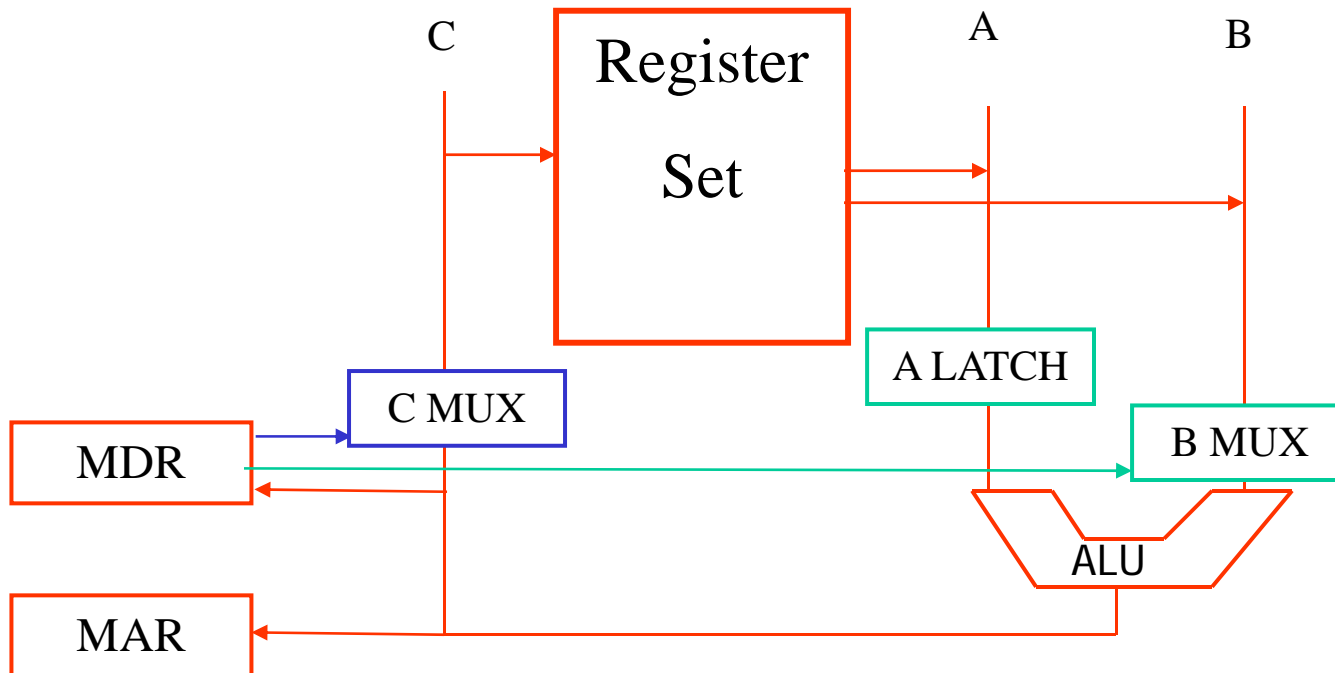
Format D: 16 inst



From Format A we need:

Mem \leftarrow R1, R1 \leftarrow Mem, R1 \leftarrow R1 OP Mem

OP R1 B2 D2

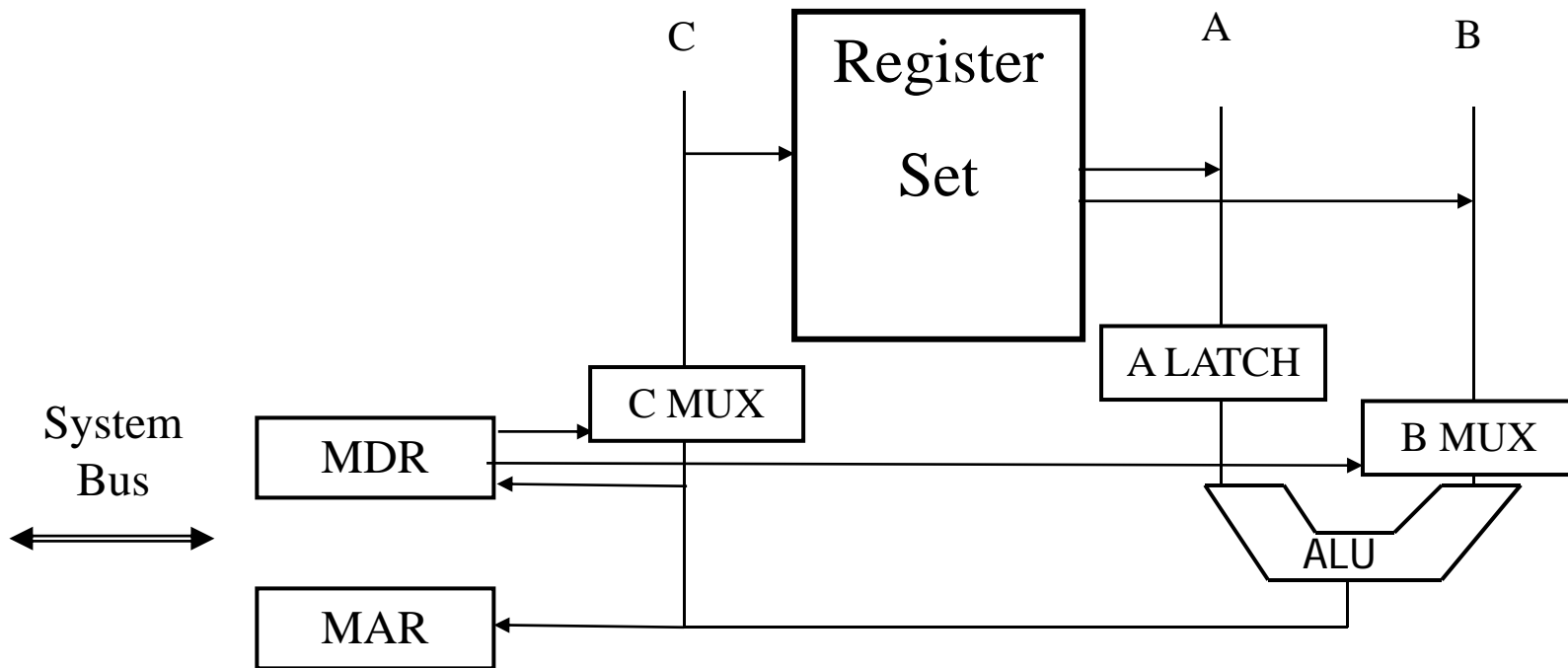


For Format B:

$R1 \leftarrow R1 \text{ OP } R2$

OP R1 R2

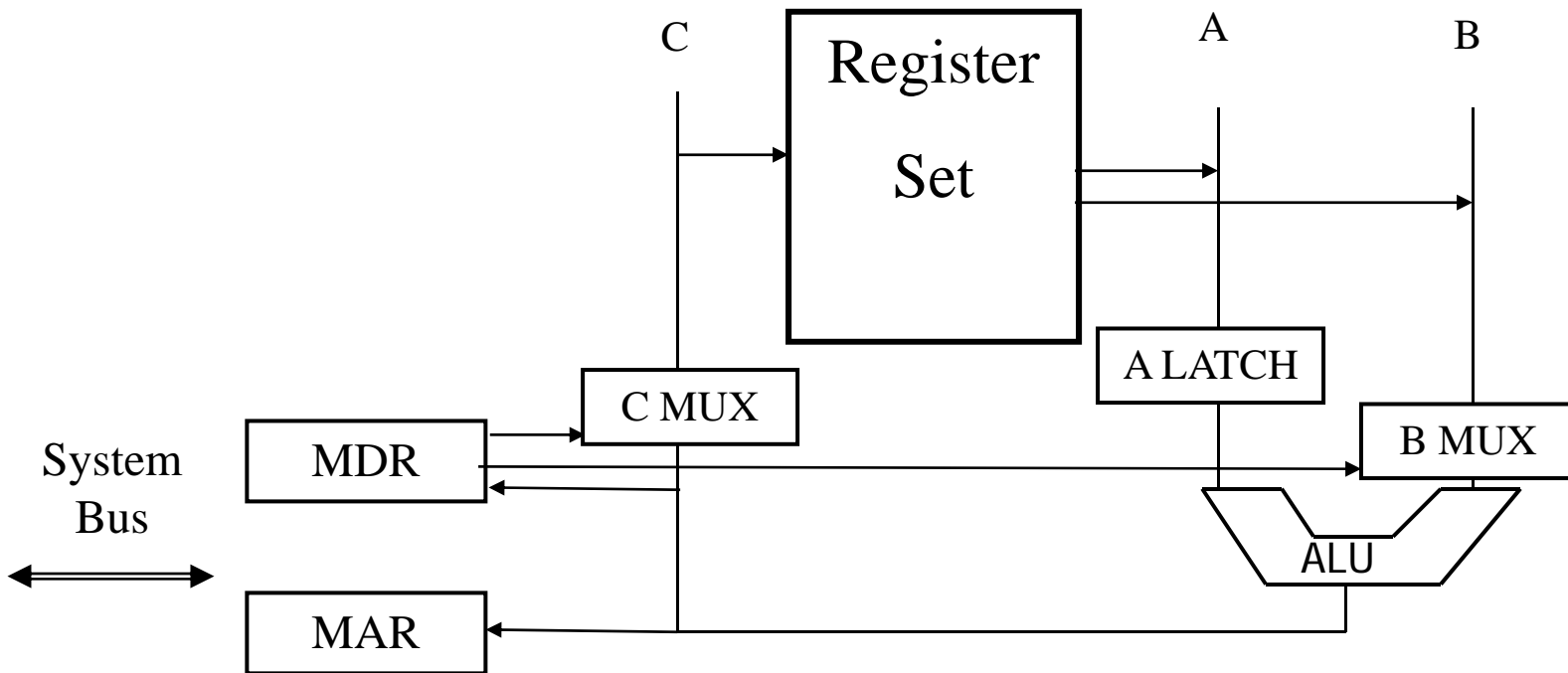
Nothing else needed



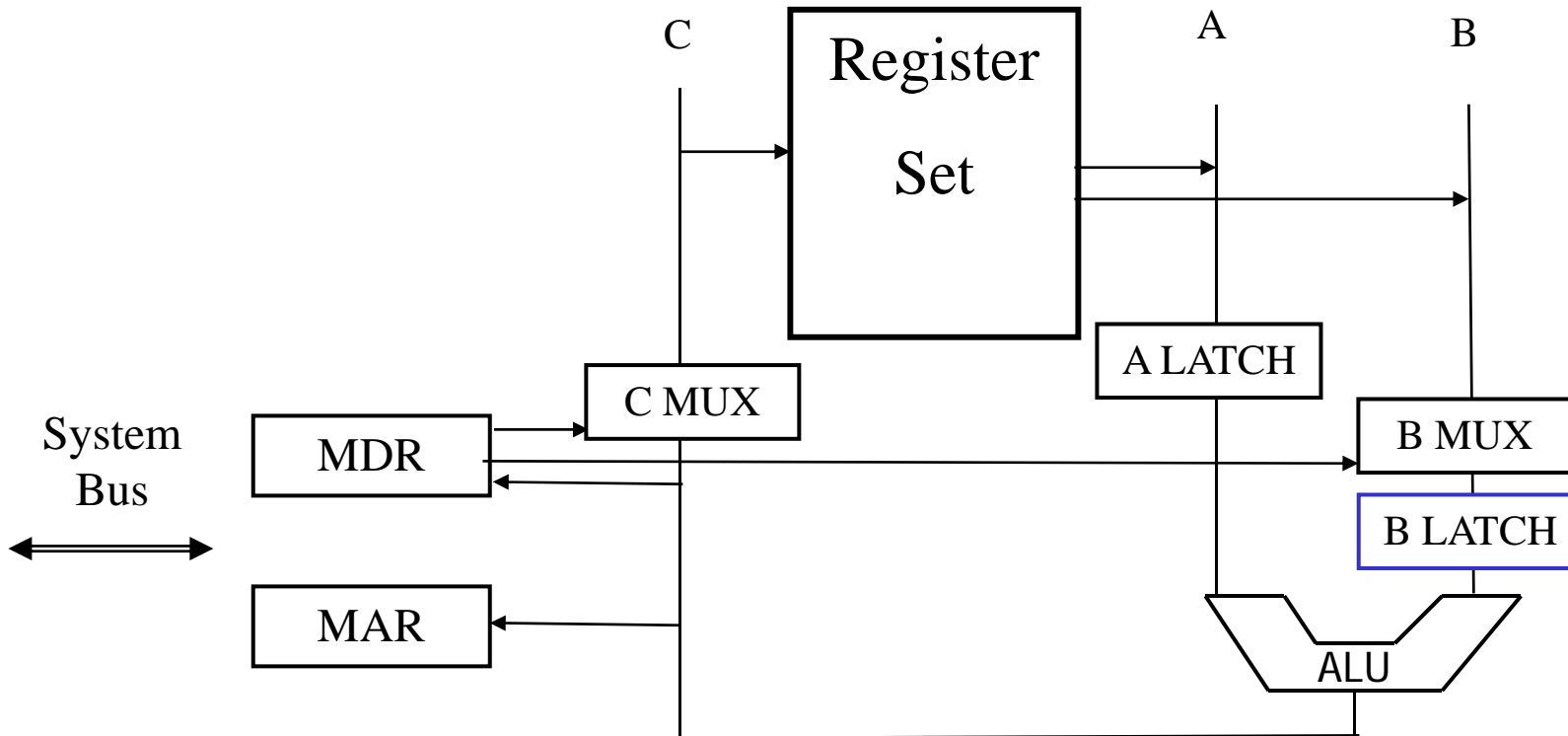
For Format C:

$R2 \leftarrow R1 \text{ OP Mem}$, $\text{Mem} \leftarrow R1 \text{ OP } R2$ OP R1 R2 B3 D3

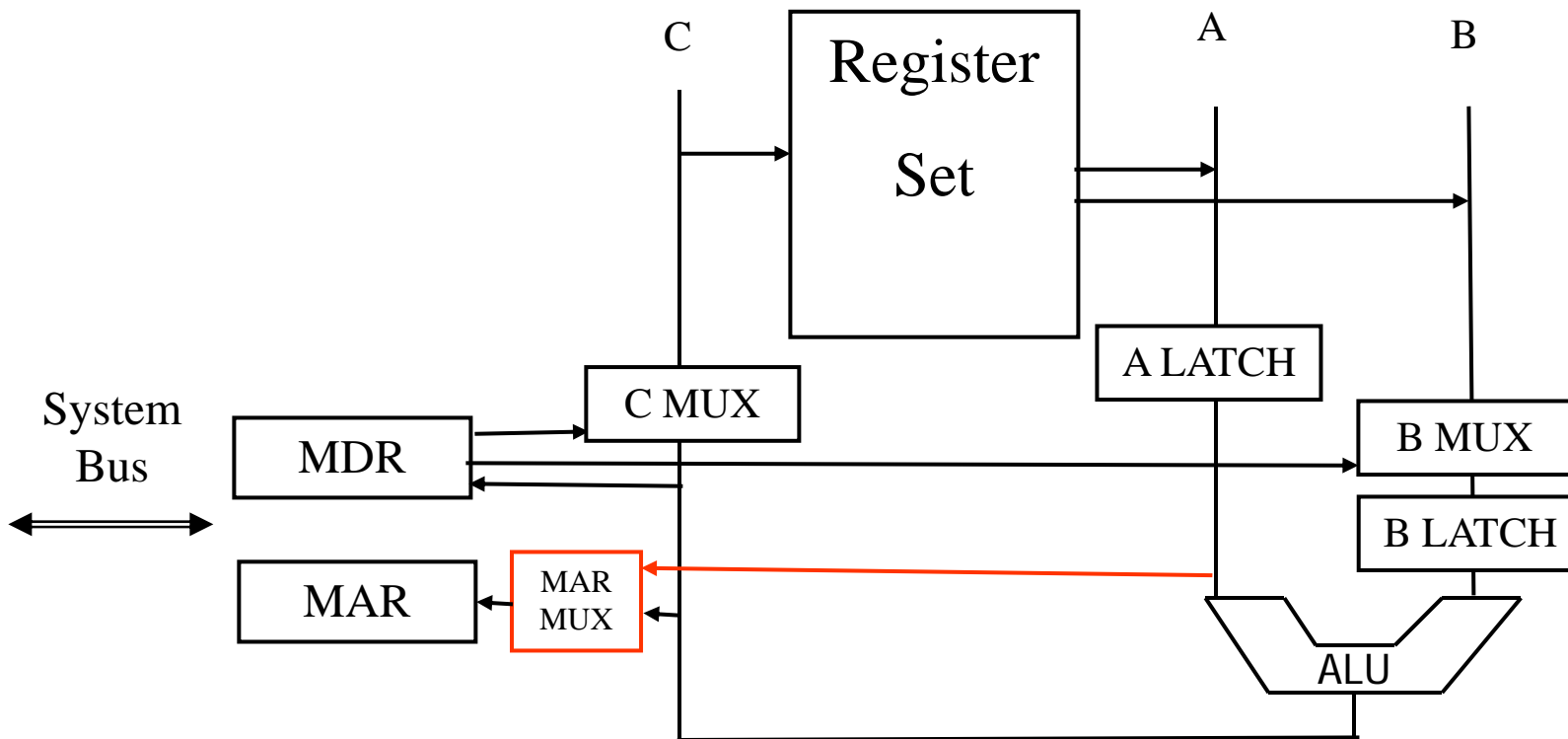
Nothing else needed.



Format D: extra: Mem2 \leftarrow R1 OP MEM1 OP R1 B2 D2 B3 D3



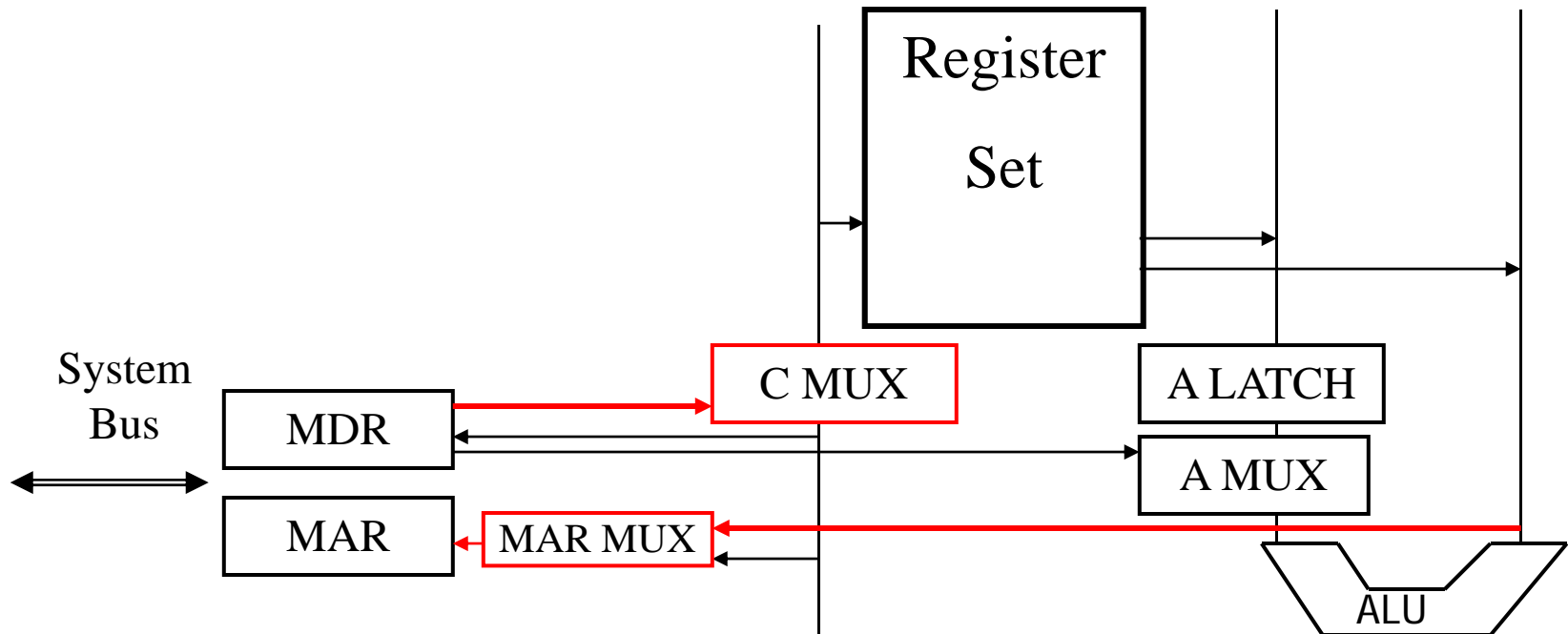
- Address of next instruction is in a register: PC
- Address must be sent to the MAR, AND be incremented and returned to the PC.



MICROCODE

- A way to implement the control functions of the control unit
- Control the buses, multiplexors, latches, registers for read/write, etc.
- Two mechanisms:
 - Hardwired – hard to do on complex machines
 - Microcode – an internal “programming language” that implements the fetch/decode/execute cycle. Machine code instructions are **interpreted** by the microcode!

- Registers to a bus: 16 G.P. + PC, IR, ? = 5 bits
- So, need 5 bits each for a, b, c bus control
- 1 bit for a-latch (0-maintain, 1-latch new)
- 1 bit for a-mux (0-a bus, 1-MDR)
- X bits for ALU control (opcode)
- 1 bit for MAR-mux (0-c bus, 1-b bus)
- 1 bit for c-mux (0-c bus, 1-MDR)
- 1 bit for MAR (0-maintain, 1-latch new)
- 1 bit for MDR (0-maintain, latch from c bus)



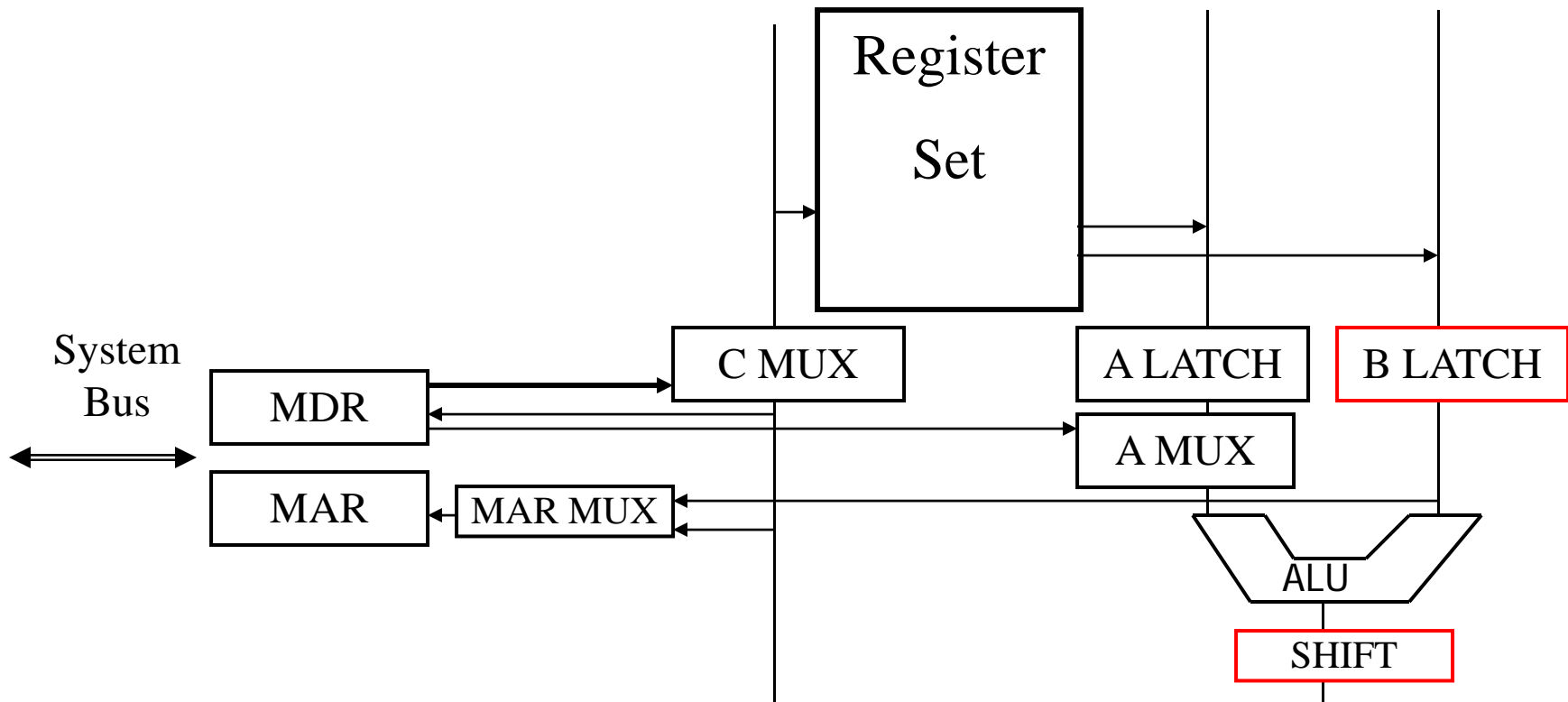
- Need to be able to alter the sequence of instructions: Jump/Branch to a different instruction (other than the next sequential)
- Allows implementation of Ifs & LOOPS
- Based on status bits that result from ALU operations:
 - N = 1 if outcome is negative
 - Z = 1 if outcome is zero
- Need to add additional bits to the microinstruction for jumps and testing, and need an address to jump to

- Jump codes:
 - ZN
 - 00 – no jump
 - 01 – jump if outcome is negative
 - 10 – jump if outcome is zero
 - 11 – unconditional jump, (rather than jump if negative or zero)
- Address to jump to: this is an address in the MICROCODE! If allow 8 bits, then allows 256 microinstructions in a MICROCODE STORE.

- Microprogrammed CPU – “machine” instructions are interpreted/translated into a set of microinstructions, which control the actions and sequence within the CPU
- MICRO-STORE – set of microinstructions that decode each “machine” instruction
- Yes, there is a micro-fetch/decode/execute sequence
- The machine code OP-CODE is an address in the micro-store, of where to begin executing the microinstructions that will accomplish the operation.

- An alternative to a microprogrammed CPU is a hard-wired CPU, where each “machine” instructions is executed by a unique set of hardware (sequence of gates that implement the logic)
- CISC – tend to be microcoded
- RISC – could be hard-wired

- Add a separate shifter, for implementing Mult and Div algorithms, add latch on B for flexibility



- Control codings:
- For each device, we need to specify the interpretation or meaning. Examples:
- AMUX
 - 0 – from A latch
 - 1 – from MDR
- CMUX
 - 0 – from C bus
 - 1 – from MDR
- MAR-MUX
 - 0 – from C bus
 - 1 – from B bus

- Condition Codes
 - ZN
 - 00 – no jump
 - 01 – jump if outcome is negative
 - 10 – jump if outcome is zero
 - 11 – unconditional jump, rather than jump if negative or zero
- ALU
 - 00 – pass through A bus, no operation
 - 01 – ADD
 - 10 – AND
 - 11 – complement
- SHIFT
 - 00 – pass through A bus, no operation
 - 01 – SHR
 - 10 – SHL
 - 11 – undefined (perhaps a rotate?)
- Etc, for each device

- Also need to specify register codings (number them as expected)
- 00000 – 01111, 0 – F
- PC: 11111
- IR: 11110
- Note that there are 14 unused codings, which could be useful for some other purpose, perhaps for hardcoding special values, like +1 or –1 for incrementing and decrementing
- +1: 10001
- 0: 10000

To fetch the next instruction, we have to do:

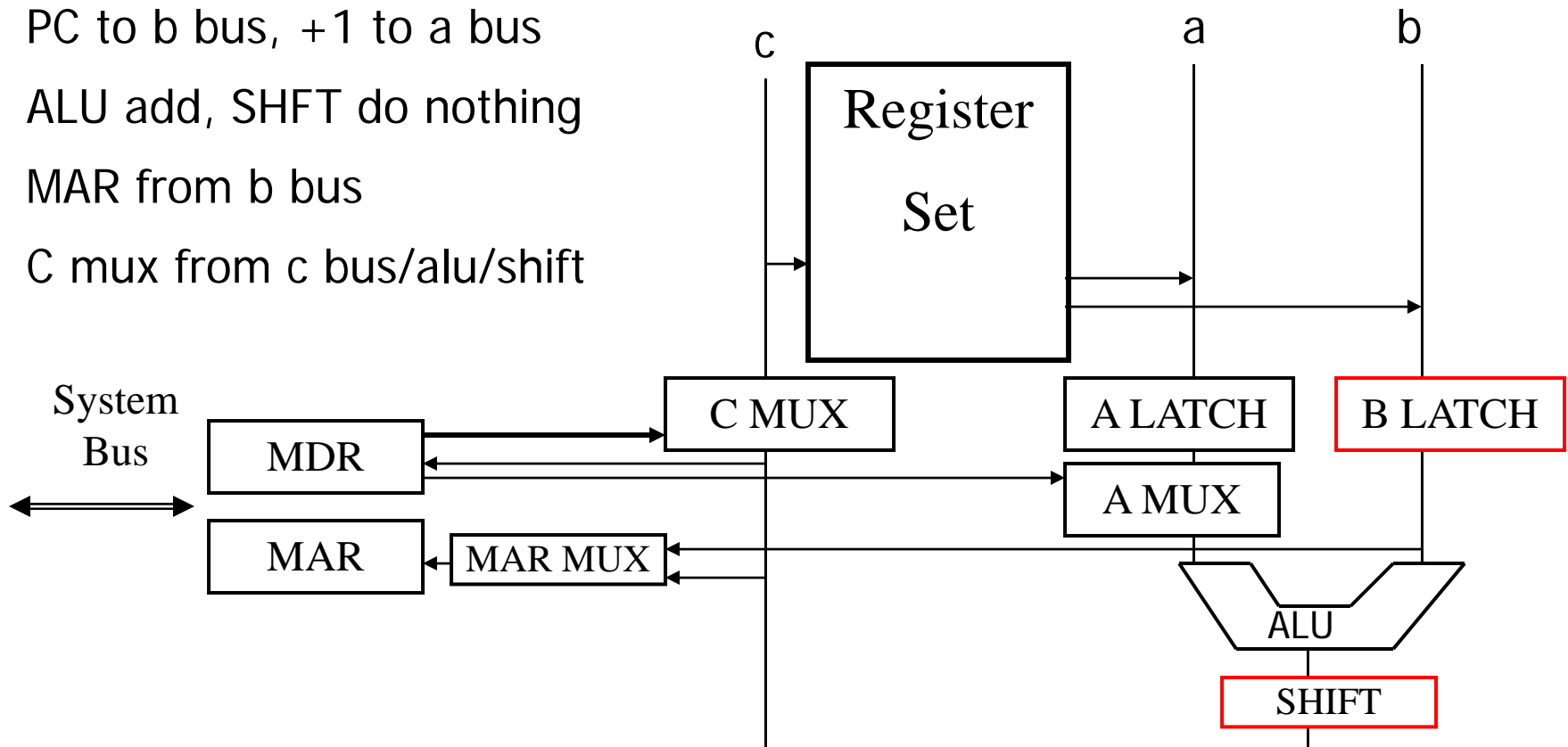
- $MAR \leftarrow PC, \quad PC \leftarrow PC + 1$

PC to b bus, +1 to a bus

ALU add, SHFT do nothing

MAR from b bus

C mux from c bus/alu/shift



To fetch the next instruction, we have to do:

- $MAR \leftarrow PC$
- $PC \leftarrow PC + 1$

muxes

A	C	MAR	COND	ALU	SHFT	MAR	MDR	A bus	B bus	C bus	micro-address
0	0	1	00	01	00	1	0	10001	11111	11111	XXXXXXXX

To AND register 1 and 2, and put the result in register 4: (fill in)

muxes

A	C	MAR	COND	ALU	SHFT	MAR	MDR	A bus	B bus	C bus	micro-address
											XXXXXXXXXX

**End
Of
Today's
Lecture.**

