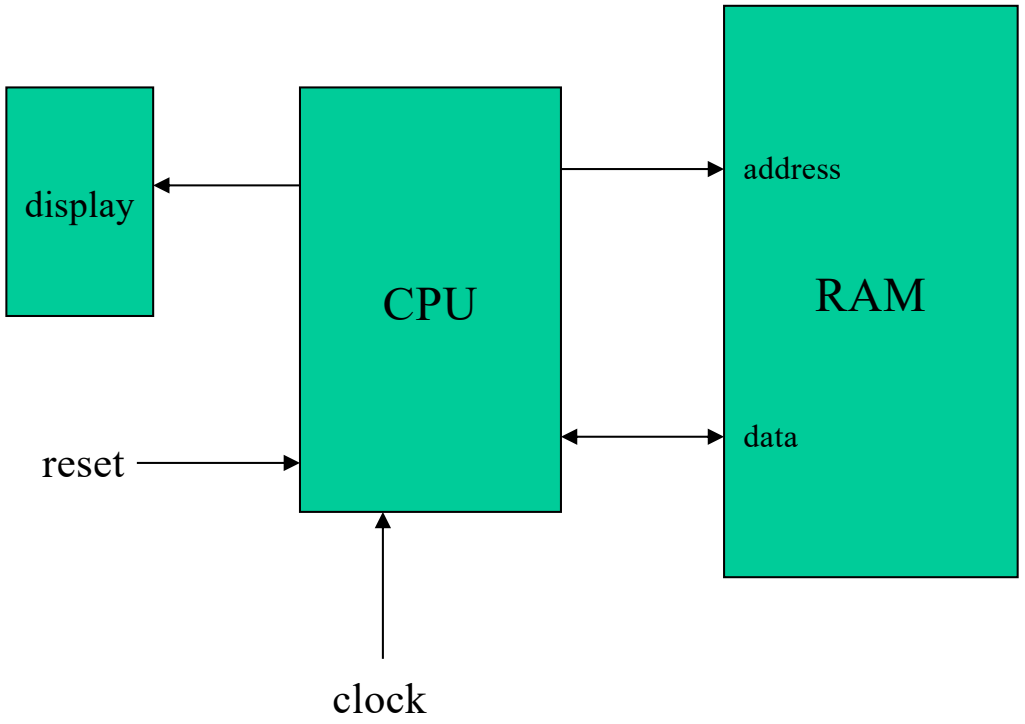
Experiment 7

- Microprogrammed CPU Design
- Implement with VHDL using lpm components and structural modeling
- Multi-week

Microprogrammed CPU

- Review Chapter 7 of Carpinelli's Computer Systems Organization & Architecture
- The entire CPU including RAM will be implemented with the DE2
- The VHDL code for the micro sequencer is provided

Overall Block Diagram



reset – set PC to 0

CPU Organization

- Program Counter (PC) – 8 bits
- Memory Address Register (MAR) – 8 bits
- Instruction Register (IR) – 8 bits
- Memory Data Register (MDR) – 8 bits
- Register File (R0-R1) – 8 bits
- Condition flip-flop (Z) – 1 bit
- Stack Pointer (SP) – 8 bits

CPU Instruction (Fall 2025)

Instruction	Instruction code	Operation
NOP	00000000	No operation
LOADI Rn,X	0001000n X	Rn ← X
LOAD Rn,X	0010000n X	Rn ← M[X]
STORE X,Rm	0011000m X	M[X] ← Rm
MOVE Rn,Rm	0100000n	Rn ← Rm, m ≠ n
ADD Rn,Rm	0101000n	Rn ← Rn + Rm, m ≠ n
SUB Rn,Rm	0110000n	Rn ← Rn – Rm, m ≠ n
TESTNZ Rm	0111000m	Z ← not V, V = OR of the bits of Rm
TESTZ Rm	1000000m	Z ← V, V = OR of the bits of Rm
JUMP X	10010000 X	PC ← X
JUMPZ X	10100000 X	If (Z = 1) then PC ← X
LOADSP X	10110000 X	SP ← X
PEEP Rn	1100000n	Rn ← M[SP]
PUSH Rn	1101000n	M[--SP] ← Rn Pre-decrement
POP Rn	1110000n	Rn ← M[SP++] Post-increment
HALT	11110000	PC ← 0, stop microsequencer

X = 8-bit data or address

m, n = 0 or 1, m ≠ n

Mnemonic	Code (Binary)	Code (HEX)
nop	00000000	00
loadi R0,FF	00010000 11111111	10 FF
loadi R1,FF	00010001 11111111	11 FF
load R0,FF	00100000 11111111	20 FF
load R1,FF	00100001 11111111	21 FF
store FF,R0	00110000 11111111	30 FF
store FF,R1	00110001 11111111	31 FF
addi r0,FF	01000000 11111111	40 FF
addi r1,FF	01000001 11111111	41 FF
add R0,R1	01010000	50
add R1,R0	01010001	51
sub R0,R1	01100000	60
sub R1,R0	01100001	61

Mnemonic	Code (Binary)	Code (Hex)
testnz R0	01110000	70
testnz R1	01110001	71
testz R0	10000000	80
testz R1	10000001	81
jump FF	10010000 11111111	90 FF
jumpz FF	10100000 11111111	A0 FF
loadsp FF	10110000 11111111	B0 FF
peep R0	11000000	C0
peep R1	11000001	C1
push R0	11010000	D0
push R1	11010001	D1
pop R0	11100000	E0
pop R1	11100001	E1
halt	11110000	F0

Instruction Format

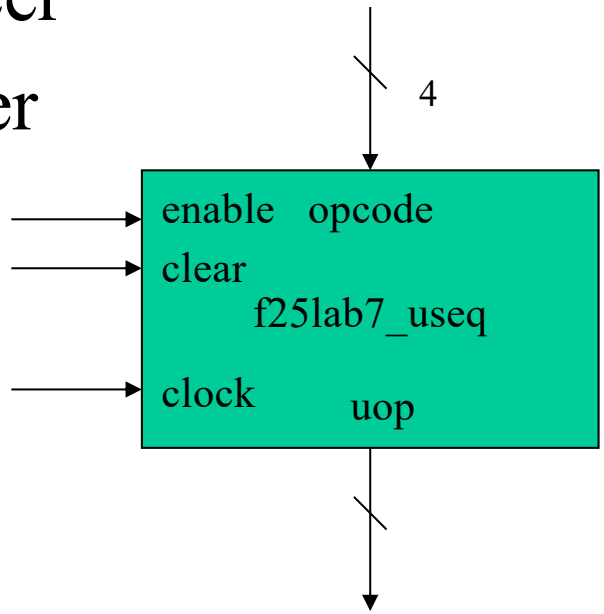
- Instructions can be 1 byte or 2 bytes
- Sixteen instructions
- 1st byte is opcode
- If present, 2nd byte is address or data

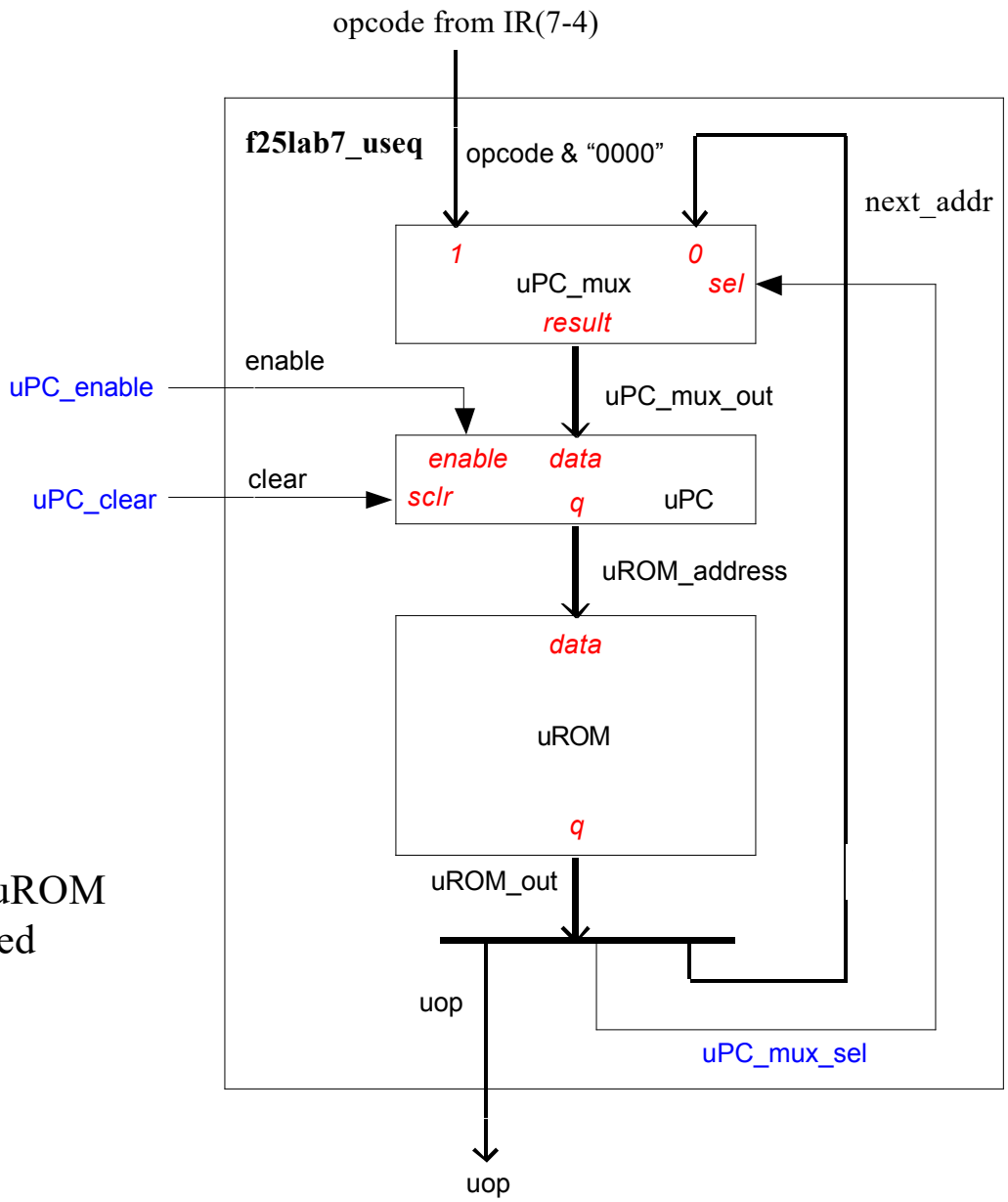
Components

- Custom Microsequencer is provided
  - The VHDL code are posted on Canvas.
- Use lpm_counter for PC, SP so you can also perform load and increment
- Use lpm_ff for other registers
- You can use VHDL statements for simple stuffs

Microsequencer

- Clock - system clock
- Enable - enables microsequencer
- Clear - reset the address register in the microsequencer to 0
- Opcode - 4-bit
- Requires two clock cycles



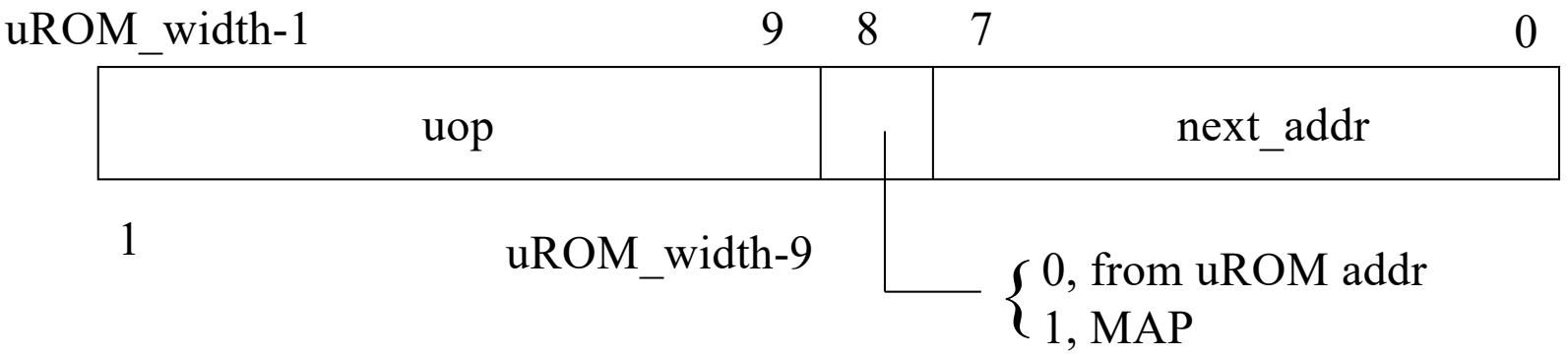


Clock signals for uROM and uPC are omitted

Micro-ROM

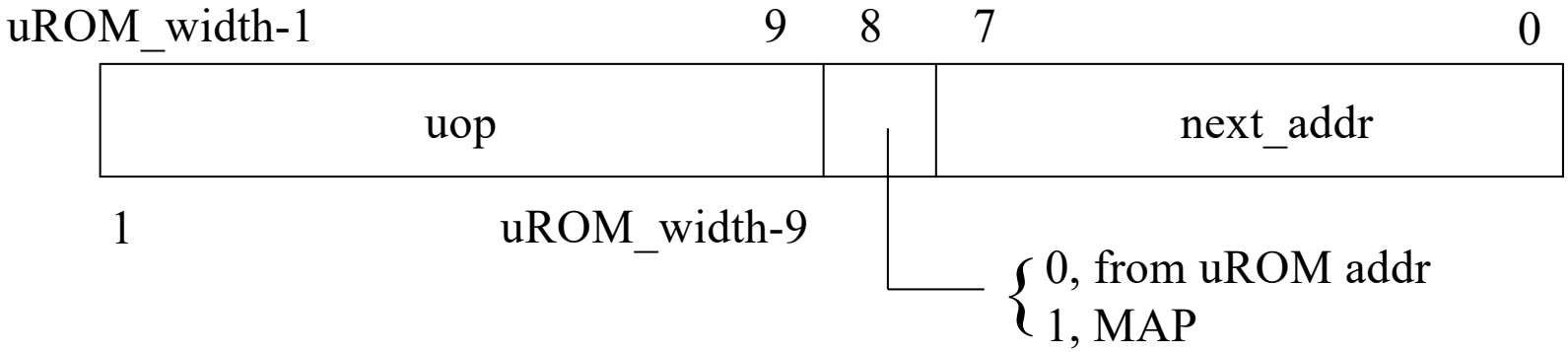
- 256 words
- Width of each micro-ROM word is uROM_width
- uROM_file is the name of the mif with the content of the micro-ROM

Micro-ROM format



Component Declaration

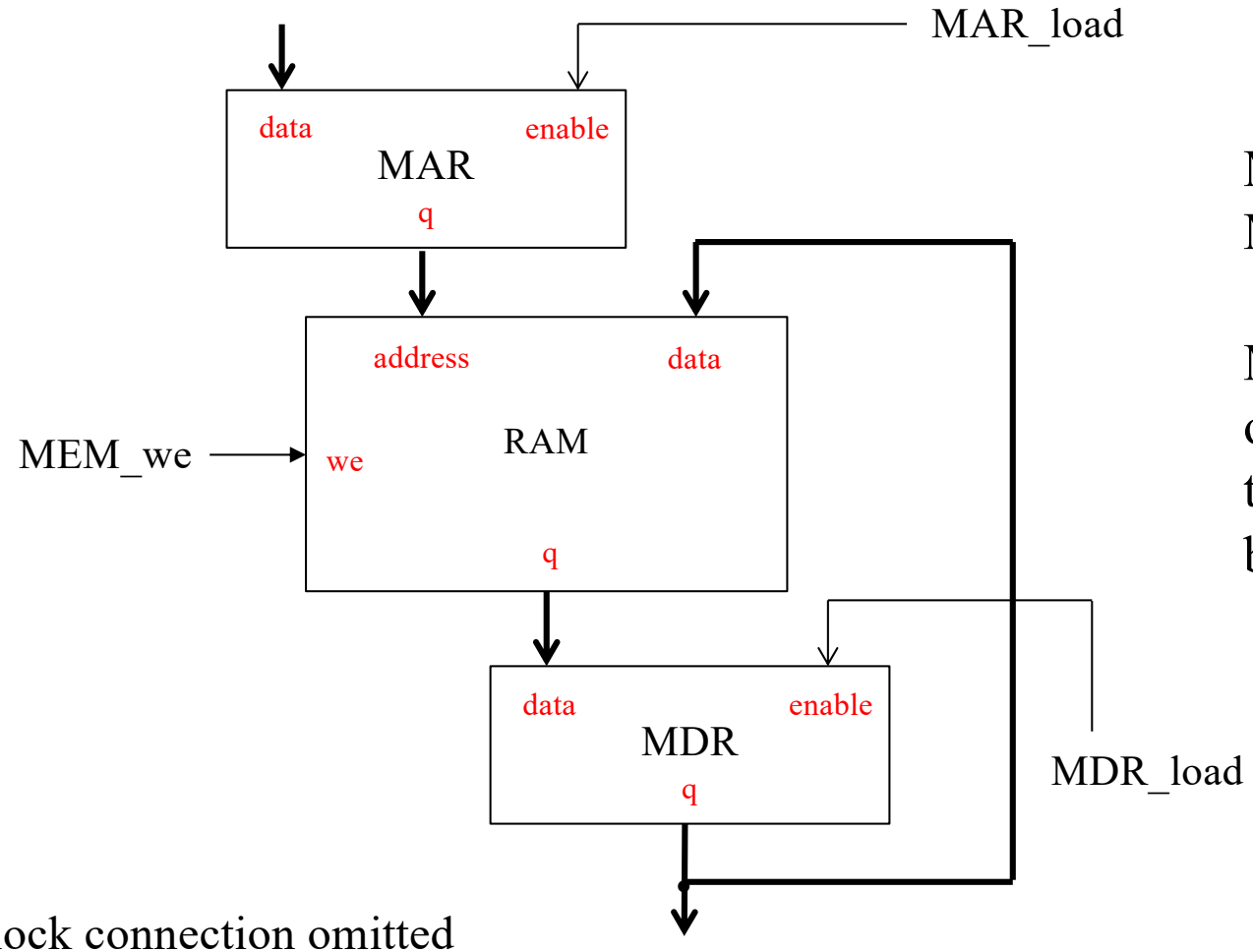
```
component f25lab7_useq
generic (uROM_width: integer;
         uROM_file: string);
port (opcode: in std_logic_vector(3 downto 0);
      uop: out std_logic_vector(1 to (uROM_width-9));
      debug_map_addr: out std_logic_vector(8 downto 0); -- for debugging
      enable, clear, clock: in std_logic);
end component;
```



Displaying Results

- Display the content of R0, R1
- Display the content of PC
- Display the content of SP
- Addition display of other registers maybe needed for debugging.
- You can use seven segment or LCD for display

RAM Connection



MAR ⟵ address
MDR ⟵M[MAR]

Memory will always output the content of the location indicated by the address port

Clock connection omitted

Register Transfer Language

- RTL is a level of abstraction that describes how data flows between registers in a digital circuit.

destination ⟵ source

Instruction Fetch

1. Move address in PC to MAR
2. Read instruction from memory and store it in MDR, increment PC
3. Move instruction from MDR to IR
4. Decode opcode in IR (done by mapping in the microsequencer)

Register Transfer Language

- Instruction Fetch
  1. MAR ← PC
  2. MDR ← M[MAR], PC ← PC+1
  3. IR ← MDR

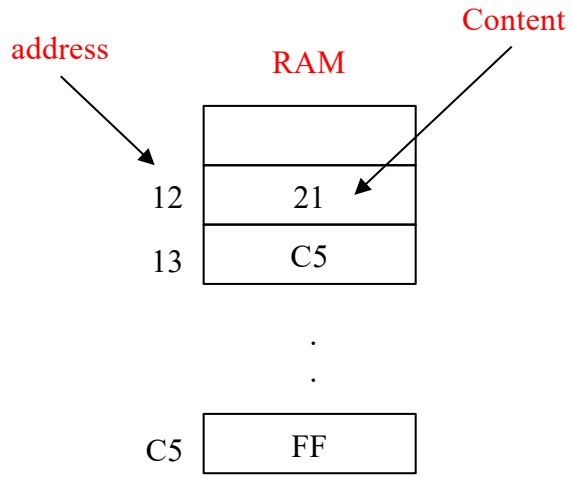
Each line corresponds to a microinstruction

LOAD Rn,X Rn ⟵ M[X]

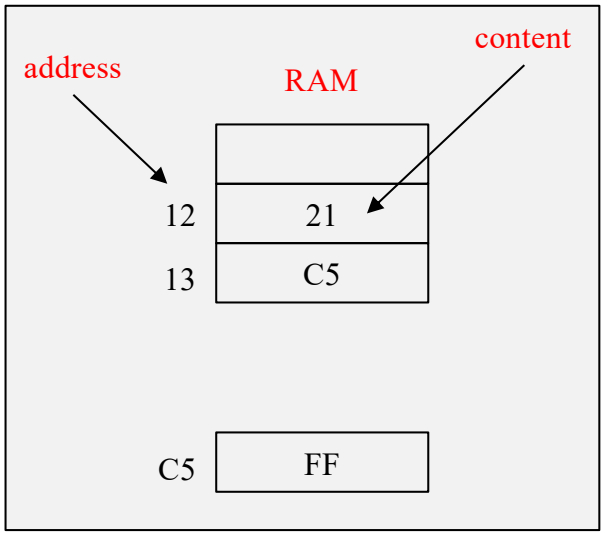
IF1: MAR ⟵ PC
IF2: MDR ⟵ M[MAR], PC ⟵ PC+1
IF3: IR ⟵ MDR

LOAD1: MAR ⟵ PC
LOAD2: MDR ⟵ M[MAR], PC ⟵ PC+1
LOAD3: MAR ⟵ MDR
LOAD4: MDR ⟵ M[MAR]
LOAD5: R0 ⟵ MDR if IR(0) = 0, R1 ⟵ MDR if IR(0) = 1

```
LOAD R1,C5 = 21 C5
           = 00100001 11000101
```



LOAD R1,C5 R1 = FF



IF1: MAR ← PC

PC	12
MAR	--
MR	--
IR	--
R1	--

PC	12
MAR	12
MDR	--
IR	--
R1	--

IF2:MDR ← M[MAR], PC ← PC+1

PC	13
MAR	12
MDR	21
IR	--
R1	--

IF3: IR ← MDR

PC	13
MAR	12
MDR	21
IR	21
R1	--

LOAD1: MAR ← PC

PC	13
MAR	13
MDR	21
IR	21
R1	--

LOAD2: MDR ← M[MAR], PC ← PC+1

PC	14
MAR	13
MDR	C5
IR	21
R1	--

LOAD3: MAR ← MDR

PC	14
MAR	C5
MDR	C5
IR	21
R1	--

LOAD2: MDR ← M[MAR]

PC	14
MAR	C5
MDR	FF
IR	21
R1	--

LOAD5: R1 ← MDR if IR(0) = 1

PC	14
MAR	C5
MDR	FF
IR	21
R1	FF

IF1: MAR ← PC
IF2: MDR ← M[MAR], PC ← PC+1
IF3: IR ← MDR
LOAD1: MAR ← PC
LOAD2: MDR ← M[MAR], PC ← PC+1
LOAD3: MAR ← MDR
LOAD4: MDR ← M[MAR]
LOAD5: R0 ← MDR if IR(0) = 0, R1 ← MDR if IR(0) = 1