Lecture 30 – Processor design: The End Game

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

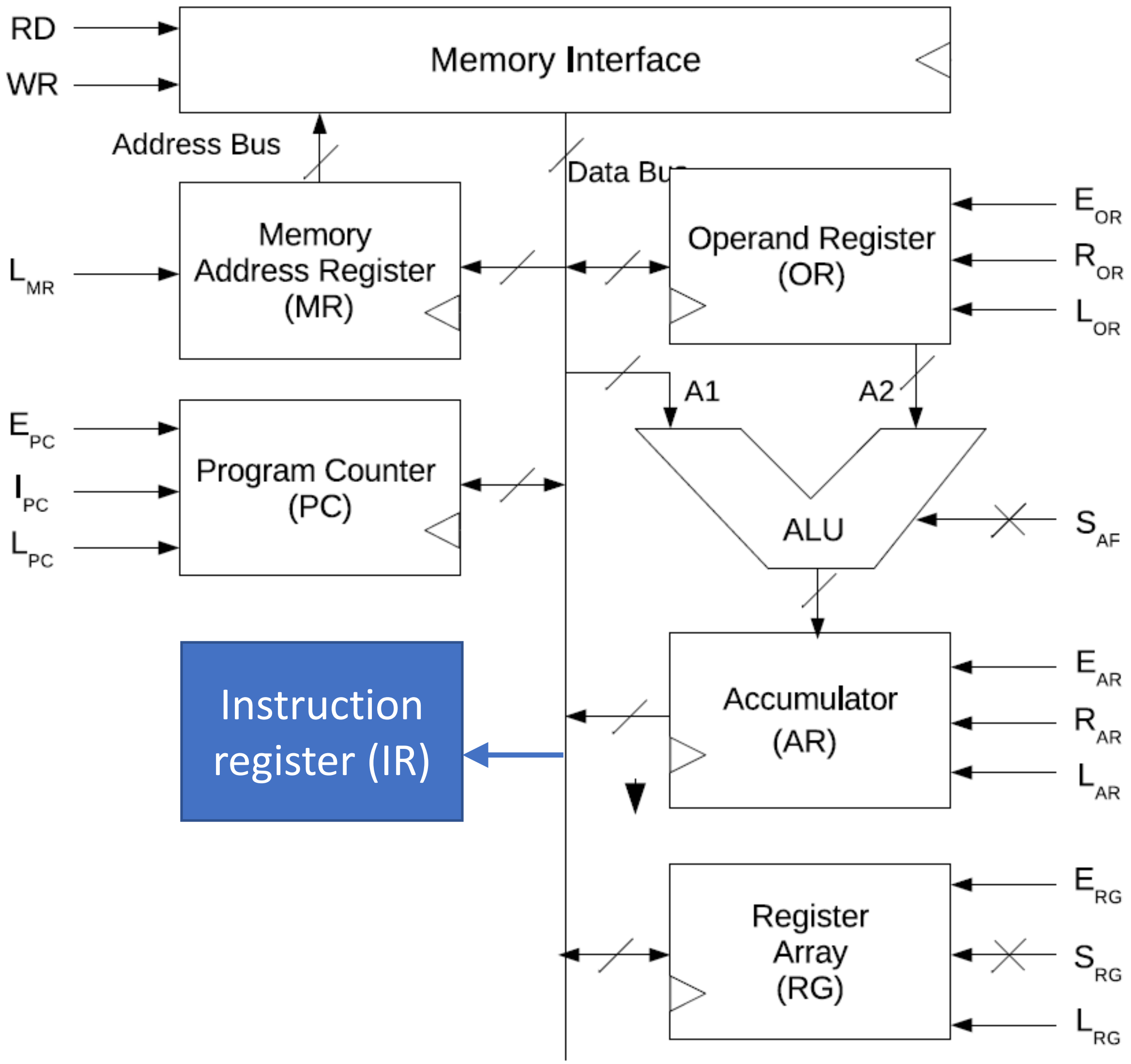
Instruction fetch

- Now let us look at instruction fetch
- We know it involves reading a word from the memory, using the PC value as the address
- This can be achieved using the following two microinstructions:

$$\text{Ck 1: } E_{PC}, L_{MR}, I_{PC}$$

$$\text{Ck 2: RD, } L_{IR}$$

- Instruction fetch requires 2 microcycles; in the first cycle, PC value is loaded to MAR
- The PC is simultaneously incremented, so that the next fetch will be from the next word in memory
- In the second cycle, the memory word at the address given by MAR is read and the value obtained is loaded into a special *instruction register* or IR
- The instruction register holds the entire opcode, which then needs to be decoded or deciphered to activate the control signals

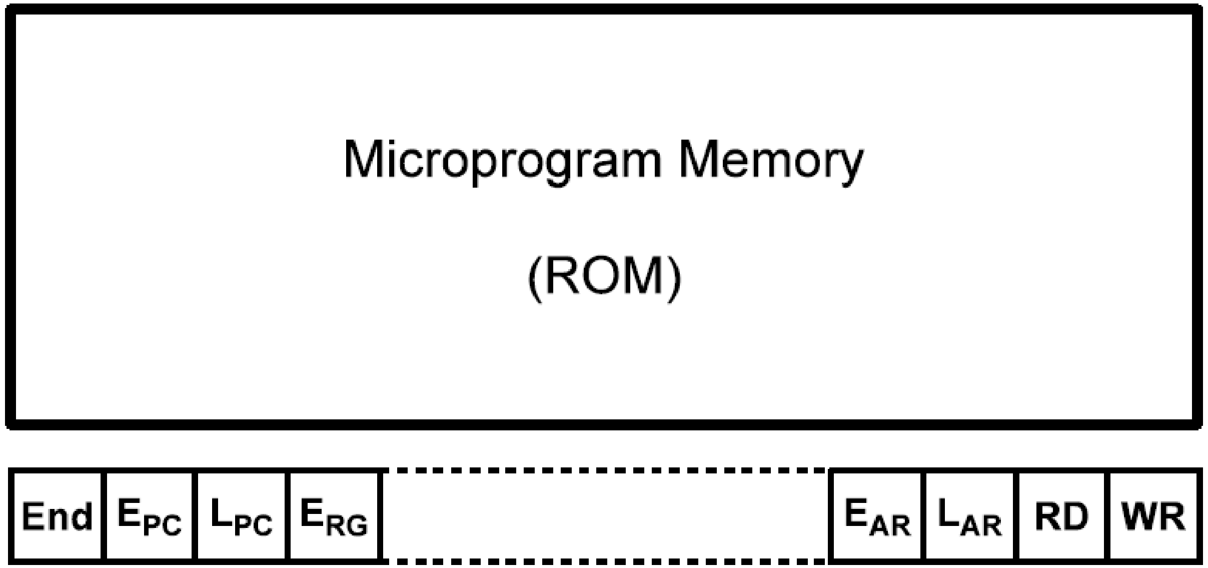


Instruction decoder

- Now, we complete the story of the processor!
- It is clear that the main task is to generate the combination of control signals for each clock cycle for each instruction as per its implementation
- We can think of the control signals being generated using a combinational circuit, whose input is the opcode of the current instruction and the clock cycle number
- The opcode is loaded into the IR register by the fetch process which needs to be decoded into the control signals
- Further, a single opcode can have multiple microcycles associated with it and all of these will have different control signals active
- The process needs to continue until the END signal line is activated

Instruction decoder

- We can use a special ROM, to generate all control signals
- The word-width of the ROM equals the number of control signals used by the processor
- The number of words in the ROM should exceed the number of distinct input combinations to have enough space to encode all the possibilities
- Each bit of the ROM output can directly serve as the control input line
- The set of control signals treated as a word is often referred to as the *microprogram* word or the control word
- Each clock cycle of each instruction maps to a separate microprogram word



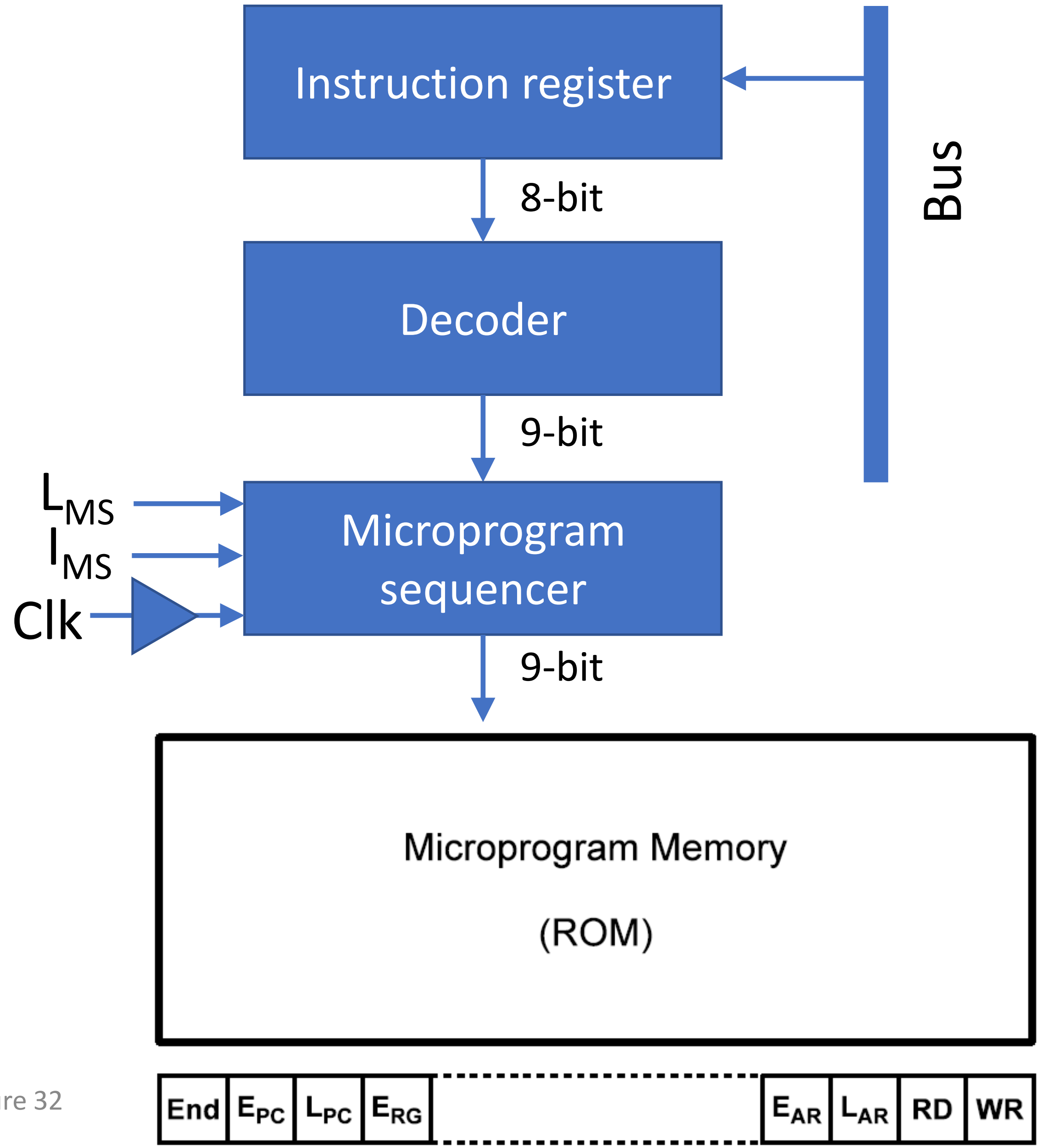
Instruction decoder

- The width of the word is clear from the number of control signals in the processor (close to 30 in this case)
- How to decide the address lines?
- The number of address lines is decided from the number of microprogram words needed to execute all the instructions
- This includes the separate microcycles within a given instruction
- For instance, ADD <R> will have two separate microprogram words associated with it
- In total, we have around 500 words, so we go with 9 address lines

Instruction	Opcode	Clk	Control Signals	Select Signals
Fetch	-	1	E_{PC}, L_{MR}, I_{PC}	-
		2	RD, L_{IR}, L_{MS}	-
nop	00	3	End	-
adi xx	01	3	E_{PC}, L_{MR}, I_{PC}	-
		4	RD, L_{OR}	-
		5	E_{AR}, L_{AR}, End	$S_{ALU} \leftarrow$ ADD
sbi xx	02	3	E_{PC}, L_{MR}, I_{PC}	-
		4	RD, L_{OR}	-
		5	E_{AR}, L_{AR}, End	$S_{ALU} \leftarrow$ SUB
xri xx	03	3	E_{PC}, L_{MR}, I_{PC}	-
		4	RD, L_{OR}	-
		5	E_{AR}, L_{AR}, End	$S_{ALU} \leftarrow$ XOR
ani xx	04	3	E_{PC}, L_{MR}, I_{PC}	-
		4	RD, L_{OR}	-
		5	E_{AR}, L_{AR}, End	$S_{ALU} \leftarrow$ AND
ori xx	05	3	E_{PC}, L_{MR}, I_{PC}	-
		4	RD, L_{OR}	-
		5	E_{AR}, L_{AR}, End	$S_{ALU} \leftarrow$ OR
cmi xx	06	3	E_{PC}, L_{MR}, I_{PC}	-
		4	RD, L_{OR}	-
		5	E_{AR}, End	$S_{ALU} \leftarrow$ CMP

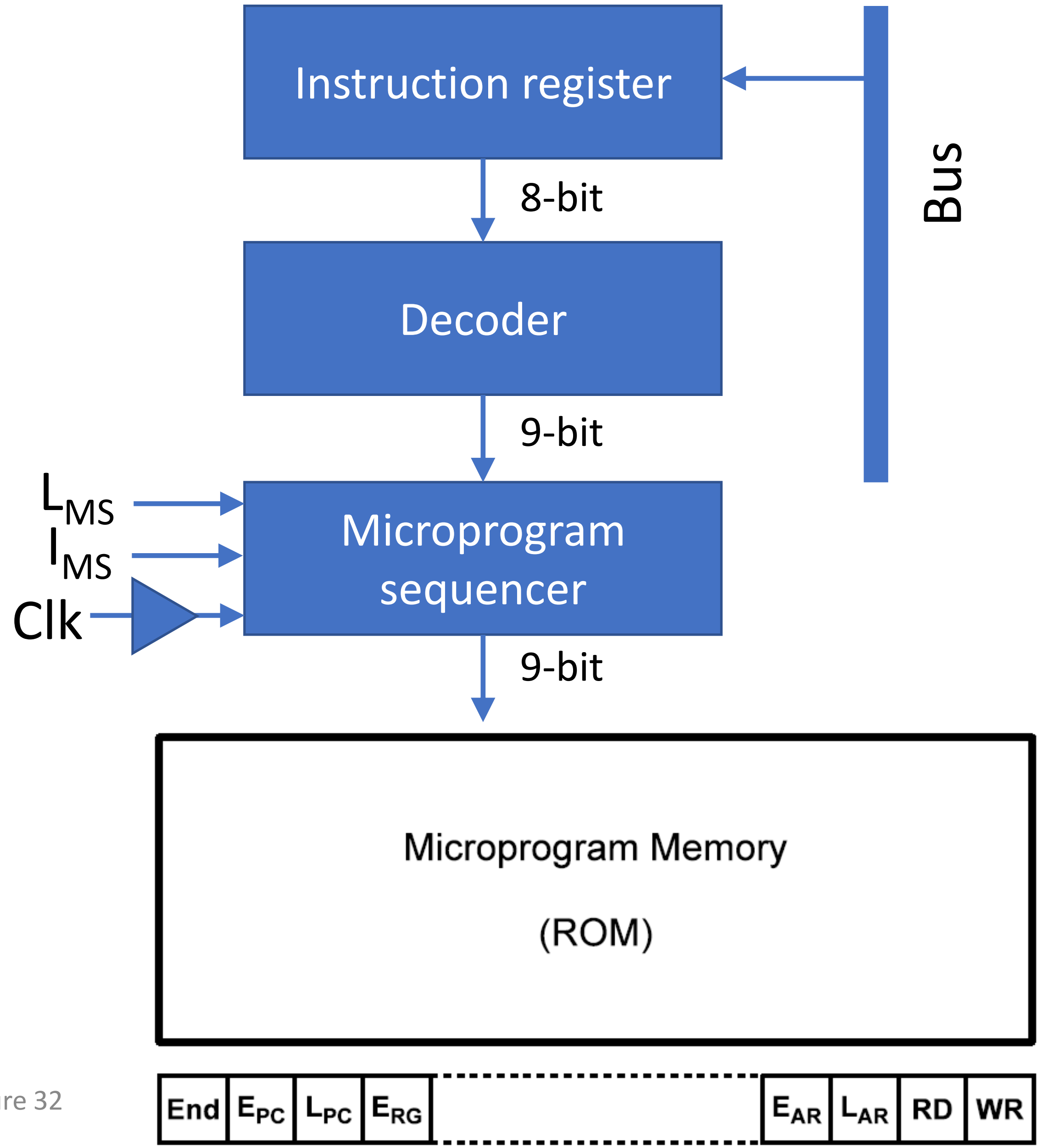
Instruction decoder

- Now, all we need to do is to map the 8-bit opcode to the 9-bit address in the ROM
- This can be done using a simple combinational circuit
- Thus, the opcode from the instruction register (IR) is decoded and stored in a register called microprogram sequencer which has an option to load it from the decoder output (L_{MS})
- Further, we store the microprogram words for a given instruction consecutively in the ROM
- The MS is incremented in every clock cycle to get the next microprogram word, until it is loaded again



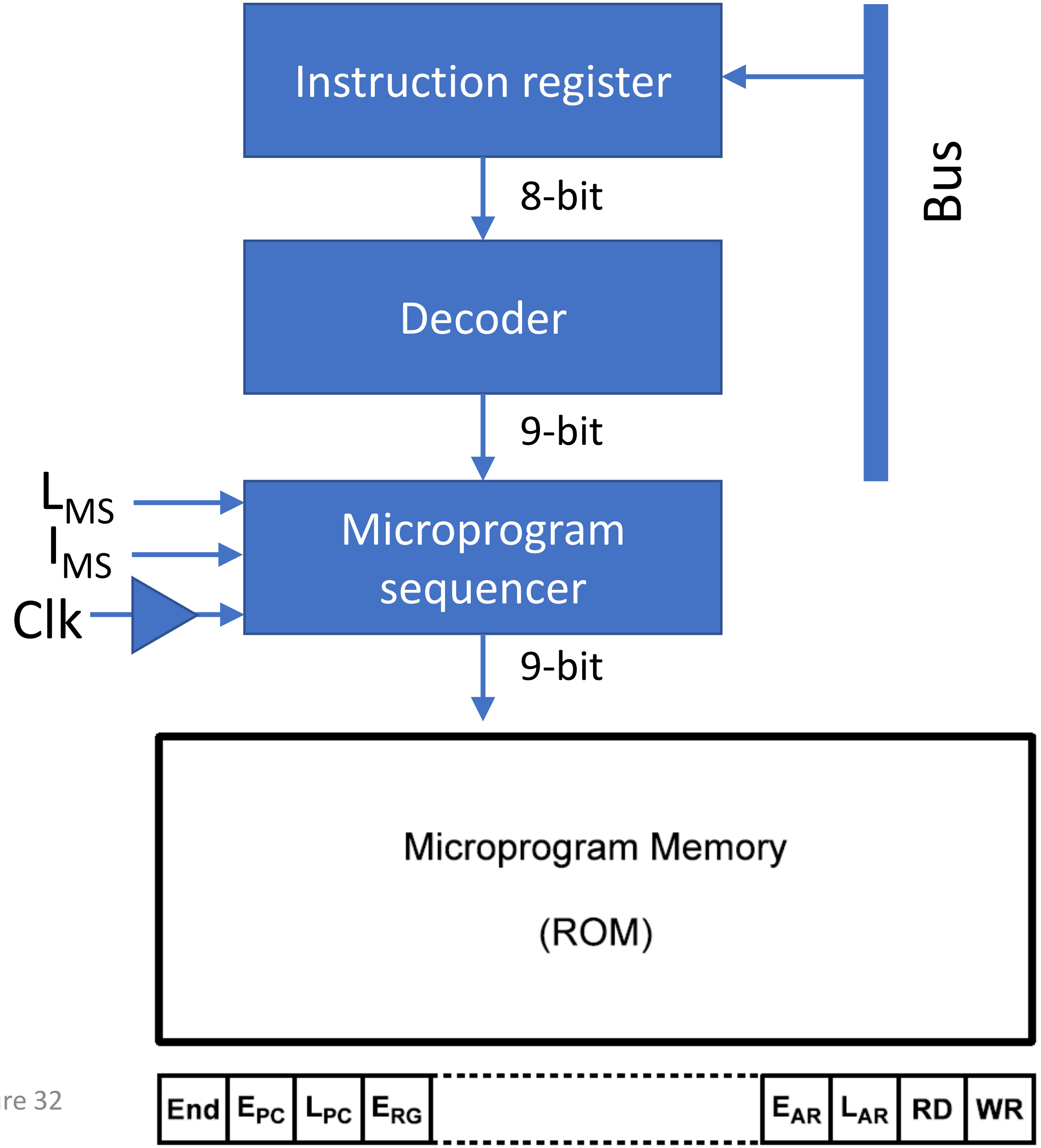
Instruction decoder

- The starting microprogram address is loaded onto the microprogram sequencer using L_{MS} (this is done using the microprogram word after the fetch cycle)
- This will result in the first microinstruction corresponding to the current opcode to be taken up in the following clock cycle
- Since we have stored the microinstructions for each instruction in consecutive locations of the microprogram memory, the microprogram sequencer is a register which gets incremented by 1 every clock cycle at the falling edge
- This continues till the last microinstruction of each opcode and the *End* line is activated



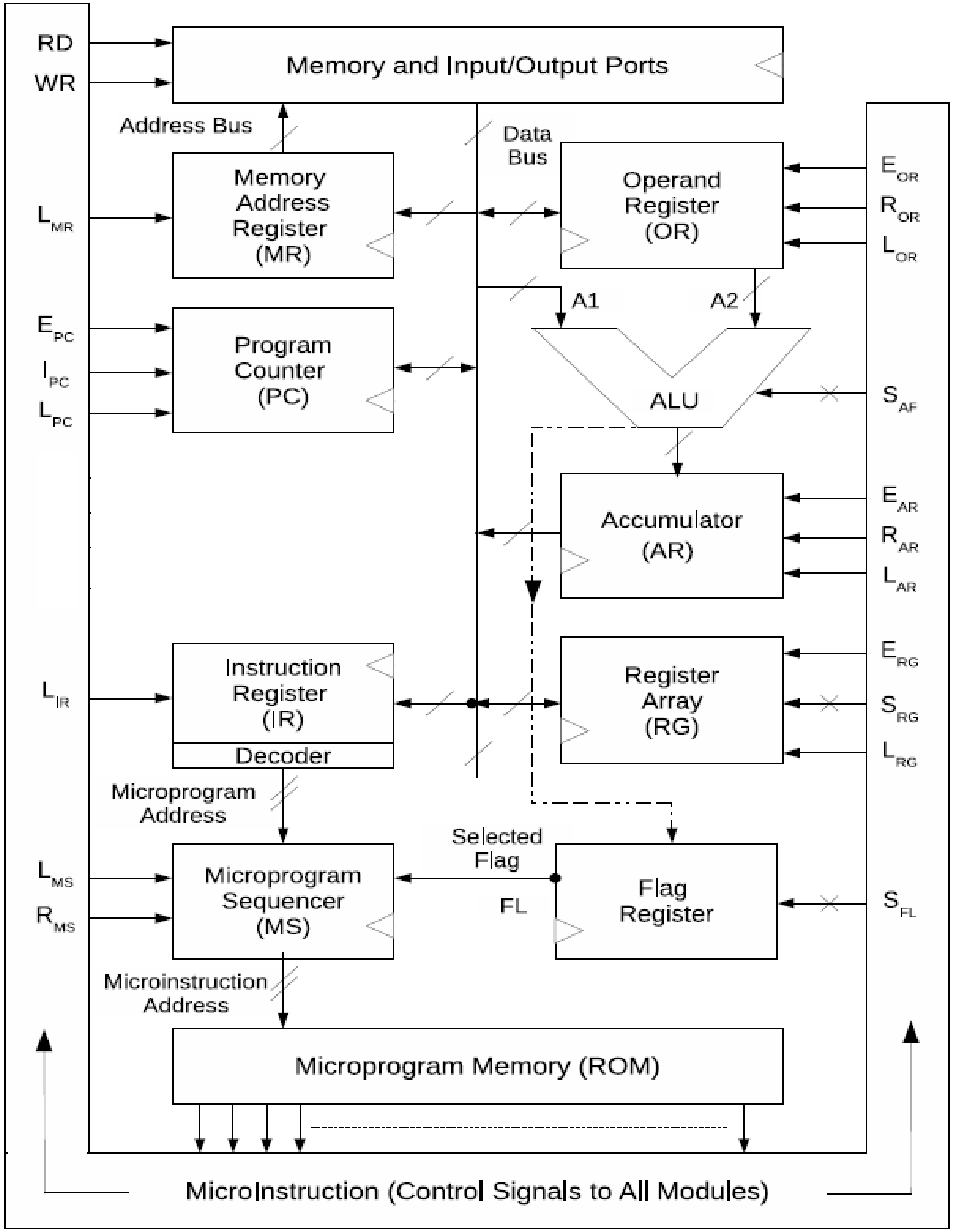
The End Game

- We issue a signal *End* to denote that the execution of the instruction is completed
- The processor is to proceed to fetching the next instruction
- How can we ensure this?
- We do this by putting the microprogram word corresponding to the first cycle of *Fetch* instruction at address 0 of the ROM
- Thus, going to the fetch phase of the next instruction can be achieved by loading 0 to the MS
- We can do that if we use the signal *End* as the reset for the MS, making it a register with load (using L_{MS}), reset (using R_{MS}), and increment facilities
- Have a power-on-reset on End, like we have for PC to reset the process to "Fetch" when the power is switched on



The final processor design

- So finally we are able to reveal the complete processor design
- This 8-bit, single bus processor can take instructions and data stored in the memory and perform tasks
- Each instruction is first fetched from memory into the IR, decoded and stored in the MS
- This is done using the address being pointed at by the PC (through the MR)
- Once, the microprogram word is obtained, the subsequent processing is done by the hardware
- Because the MS updated at the (delayed) negative edge of the clock, the control signals are available just after the negative edge , i.e., just before the positive edge



A familiar architecture

- This is the internal architecture of ATMEGA328, which is the brain of the Arduino board

- Very similar to the processor we designed, but many key functions are added such as interrupts, timers, ADC to enable overall functionality – TBD in CCIoT (UG2 ECE elective)