Lecture 29 – Processor design 4

The fetch-execute cycle

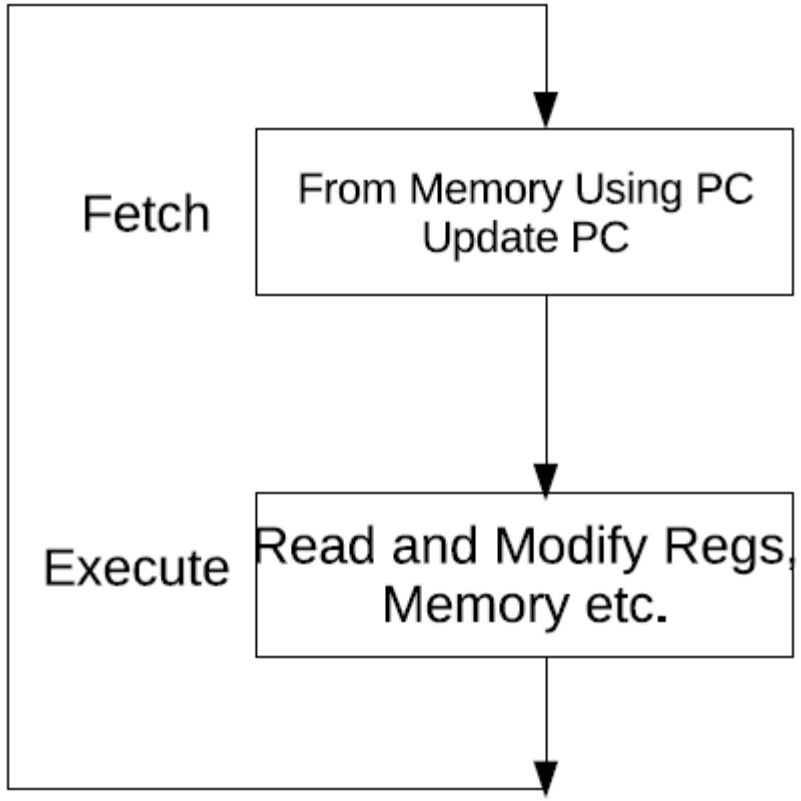
- We will look at the process of instruction fetching and execution
- The processor works autonomously as a continuous fetch-and-execute engine, with no other input than an external clock
- Since instructions as in the machine code are stored in memory, they have to be brought to the processor one by one and executed
- The instruction at address $(i + 1)$ has to be fetched and executed after instruction i, since the instructions of a program are stored consecutively in the memory
- The processor has to do all these by itself

The fetch-execute cycle

- Processors have a special register inside them that manages the process of instruction fetch by keeping track of the address of the next instruction to be fetched at all times
- This register is called the program counter or the PC
- The processing of an instruction begins with fetching its opcode from the memory word whose address is in the PC
- The contents of the PC are incremented while this happens to hold the address of the next instruction in the sequential order
- The opcode is brought to the processor and appropriate action is performed in the execution phase
- Once this is completed, the next instruction is processed by fetching it from the memory using PC as the address
- This goes on for ever inside the processor until a special STOP instruction is encountered
- Executing this instruction stops all activities of the processor

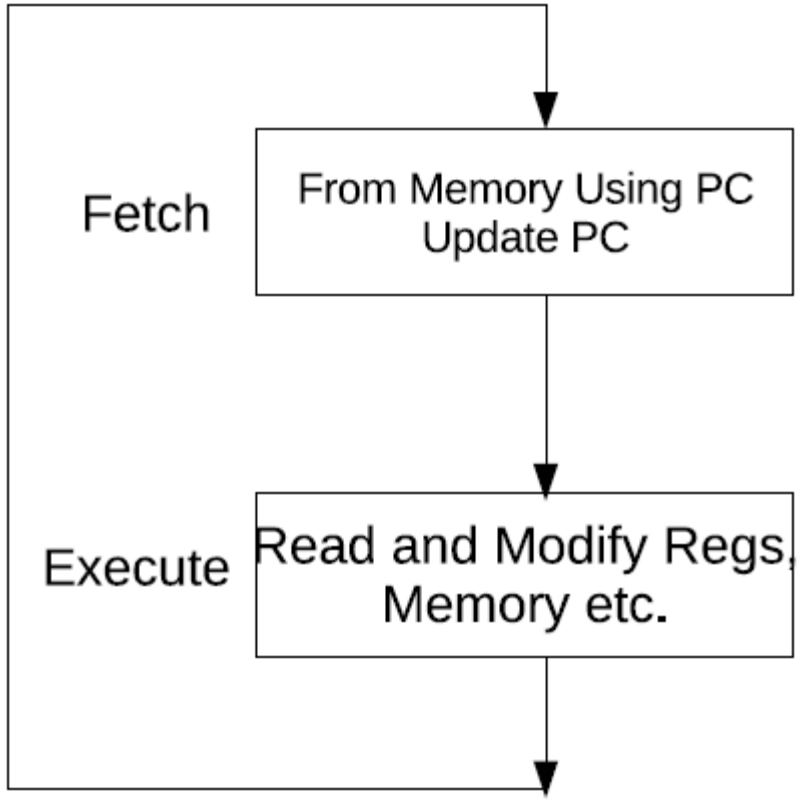
The fetch-execute cycle

- So how do we start the process?
- It is clear that once one instruction is done with, the next one is taken up by incrementing the PC
- Thus, once the execution of a program starts, everything goes on as the program indicates
- So how to start a program?
- A program can be started by loading the address of its first instruction into the PC
- However, how does the very first program start when the computer's power is turned on?



The fetch-execute cycle

- We know Operating System (OS) is the program that controls our computer
- The OS itself is loaded into the processor's memory from the hard disk on boot up prior to taking over the system
- Which program loads the operating system? How does that program get the control at the very beginning?
- Modern PCs have a program called the BIOS (Basic Input Output System), which is the very first one to get control of the processor
- How does the BIOS get control?

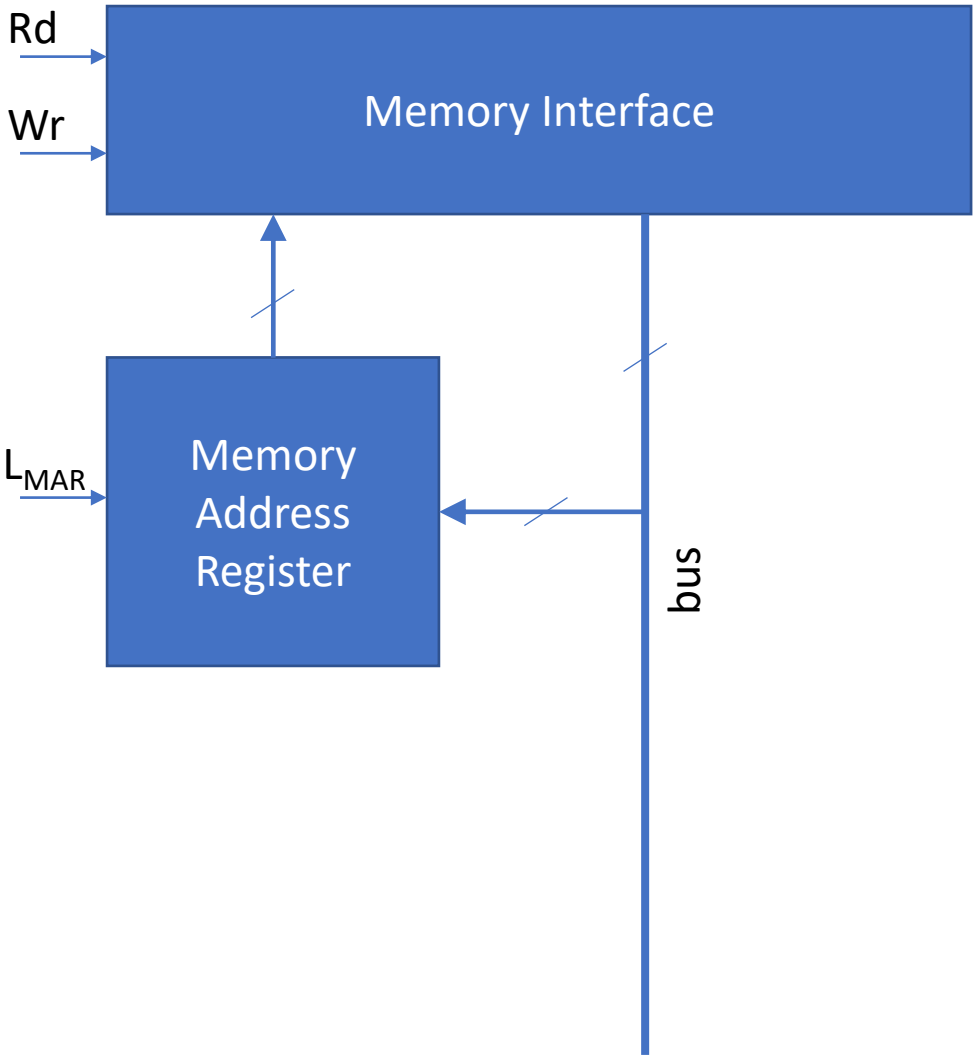


The fetch-execute cycle

- The processor hardware has a special feature to load a value of 0 to the PC when power is turned on or when the reset button of the computer is pressed (*literally* “resets” the “PC”)
- Thus, the very first program that gets control is the one that is saved at memory address 0
- Computer manufacturers place a special program at address 0 that has the BIOS program, which knows how to load the operating system from the boot record and proceed accordingly
- Any corruption in the BIOS can be very detrimental to booting the computer

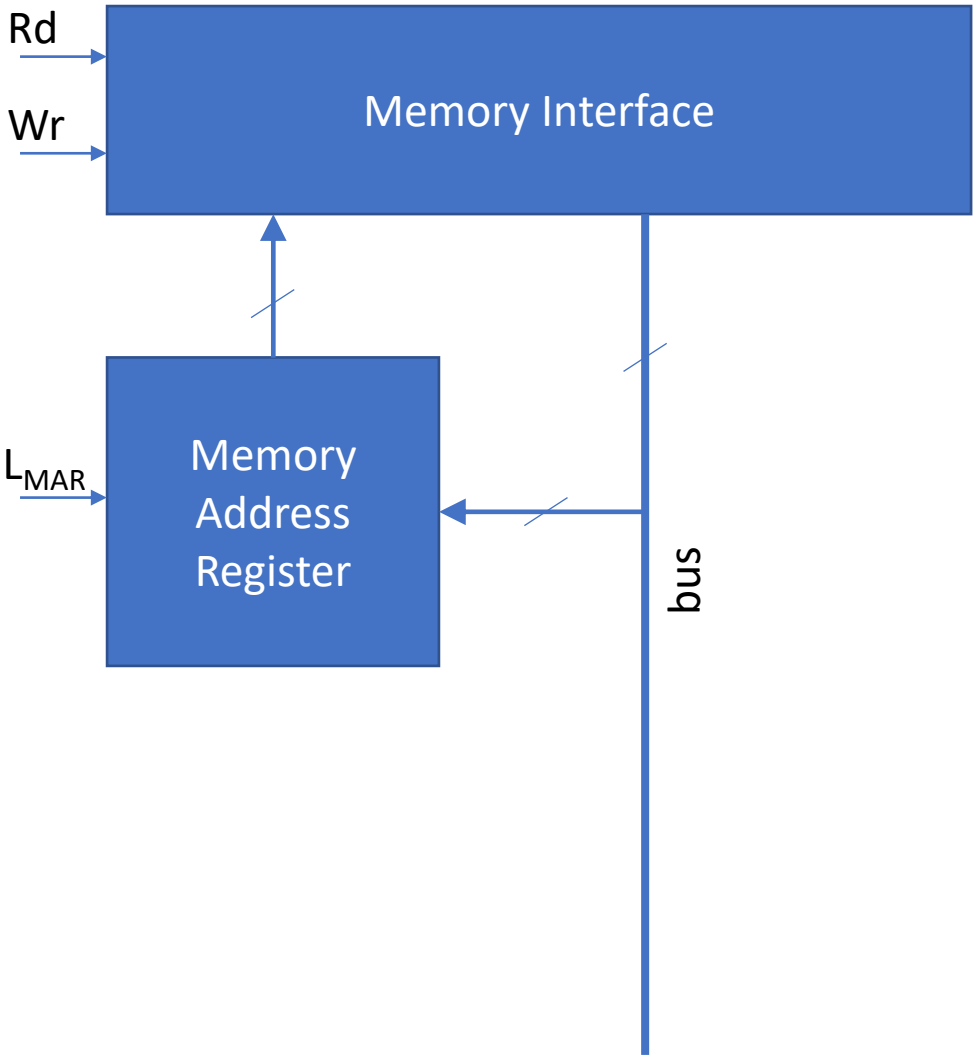
Memory access

- How does the simple processor access the memory?
- We will use a memory interface that supplies the address to the memory along with the signals to indicate if a read or a write is desired
- Data should be presented separately for writes; data supplied by the memory should be used inside the processor for reads
- We assume an external memory interface consisting of address lines, data lines, and two control lines
- The data lines are connected directly to the data lines of the bus, as if the memory is a large register array, but outside of the processor



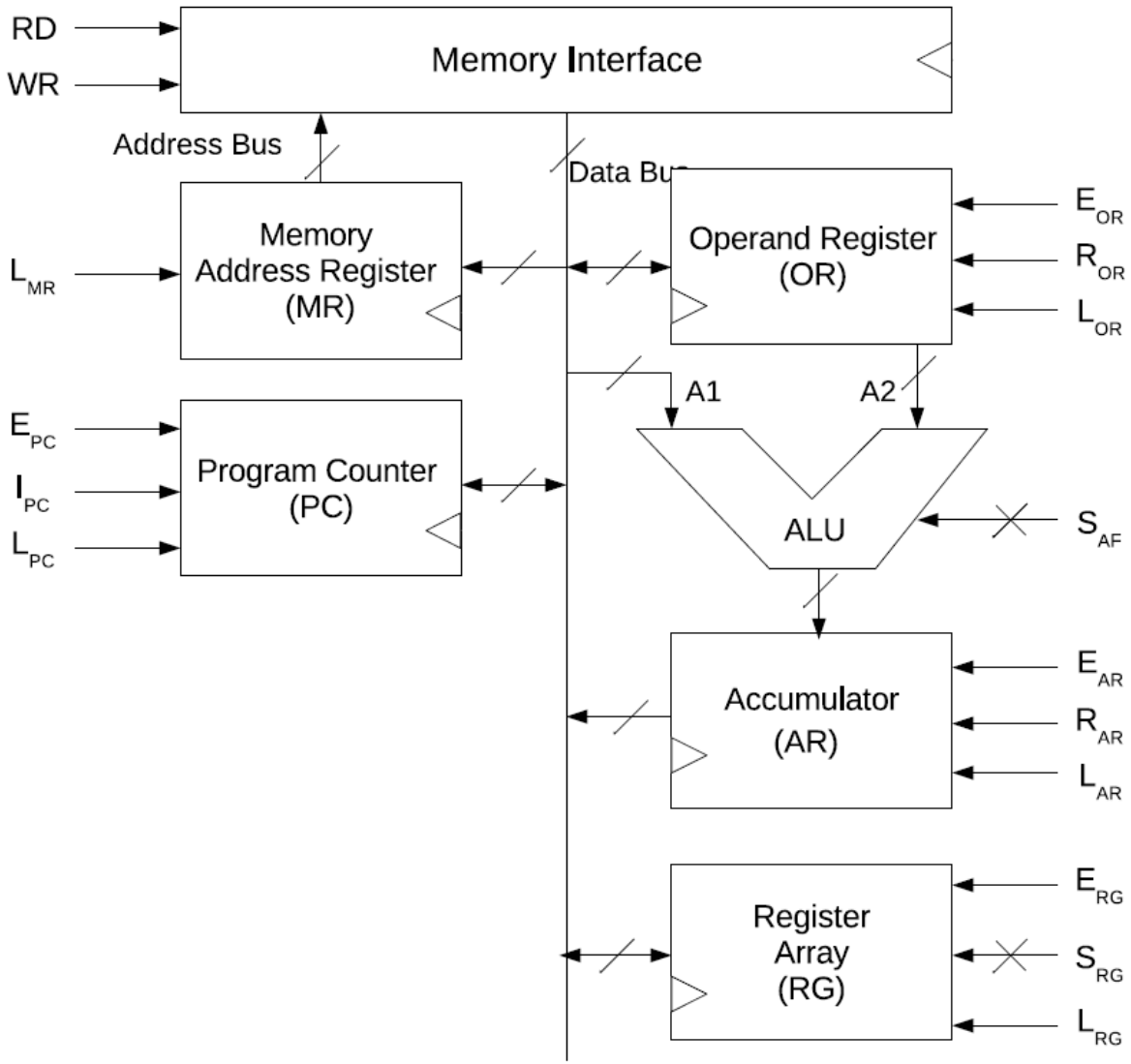
Memory access

- The address has to be supplied separately, prior to the read or write operation
- We assign a memory address register (MAR) to hold the address
- The MAR is connected to the bus like other registers and can be written to from the bus
- There is usually no need to enable the MAR to the internal bus
- It can be assumed to be enabled always to the external memory interface
- Two control lines RD and WR are sent to the memory to indicate memory read and write respectively



Enhanced enhanced single bus architecture

- With this information, the enhanced single bus architecture is modified to include additional components
- The memory address register to store the next memory address to be accessed
- The program counter to store the current address of the instruction being performed



Two more instructions

- We will introduce two more simple instructions: NOP instruction for no operation and STOP instruction to stop the processor.
- NOP does nothing: when it is executed nothing at all changes in the processor or memory.
- It may seem superfluous, but comes into use when nothing is required from the processor other than spending the required clocks to fetch and execute this instruction.
- The STOP instruction terminates the endless fetch-execute cycle that the processor is engaged in.
- The processor enters a state of complete inaction when the STOP instruction is executed.
- There is no way to come out of this state through a program as the processor has stopped looking at programs!
- Thus, the only way to come out is a hard reset through a reset button or through power cycling.

Assembly Instruction	Machine Code	Action
`nop`	00	-
`stop`	07	(Stops fetch)