Instruction Cycle and Pipelining

12.3 Instruction Cycle (pp. 423-427)
12.4 Instruction Pipelining (pp. 427-441)

Recall from Chapter 3 that the instruction cycle looks like:

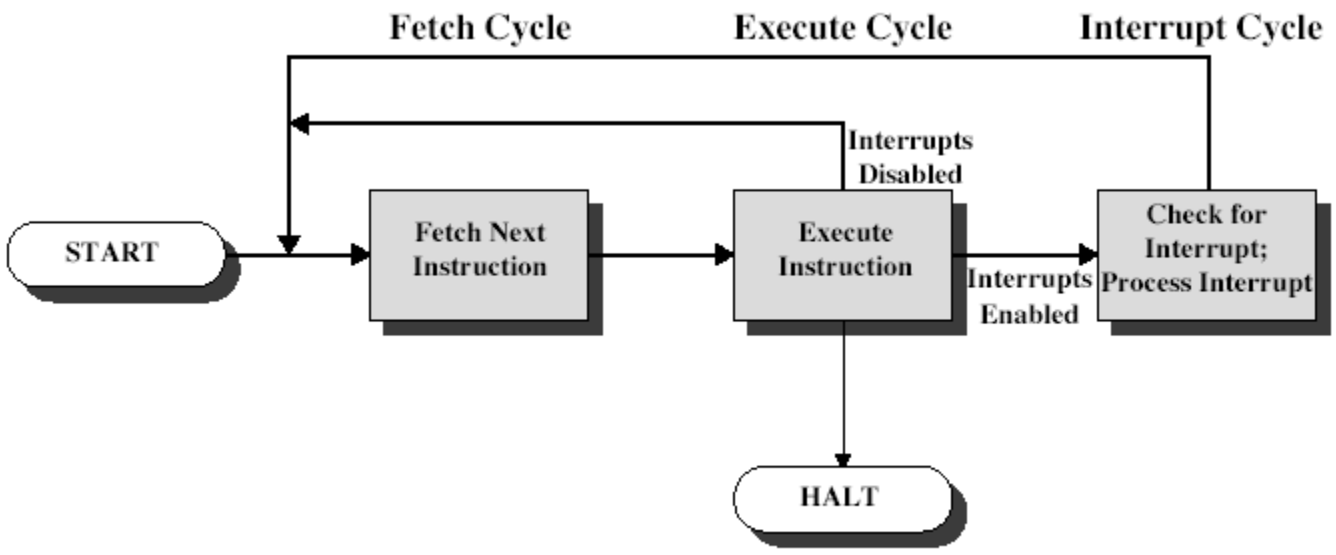
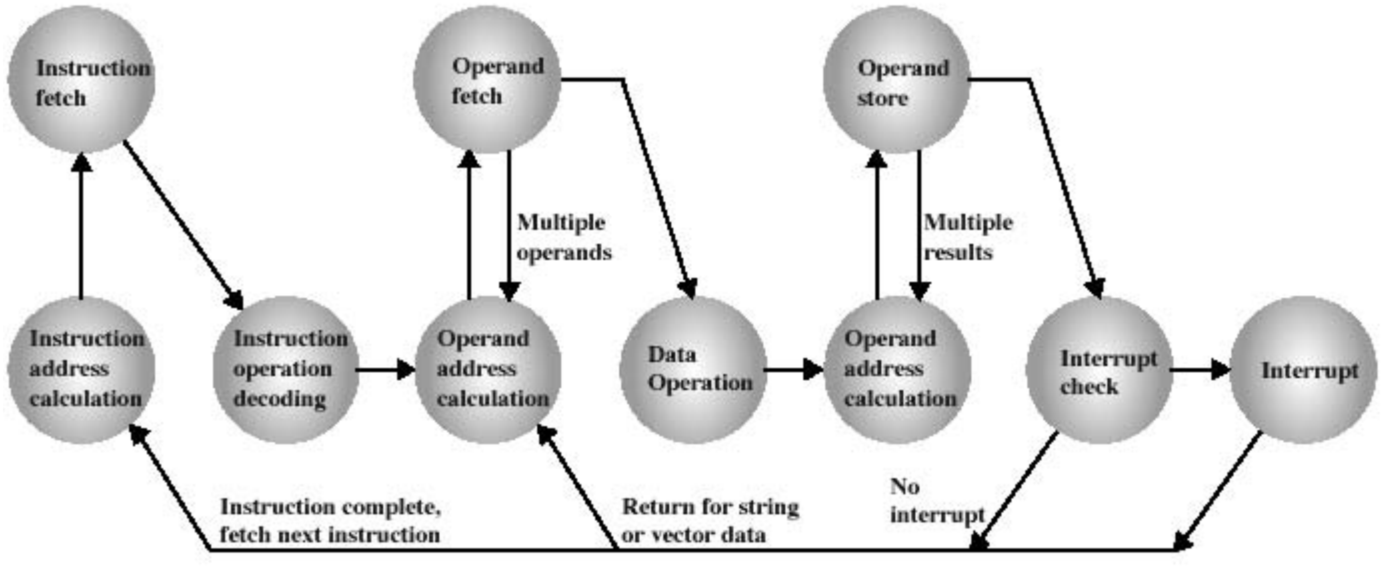


Figure 3.9 Instruction Cycle with Interrupts

We took a little more detailed look at the above diagram this way:



Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

Figure 3.12 Instruction Cycle State Diagram, With Interrupts

And after Chapter 11, we see that the complexity of the diagram now looks like:

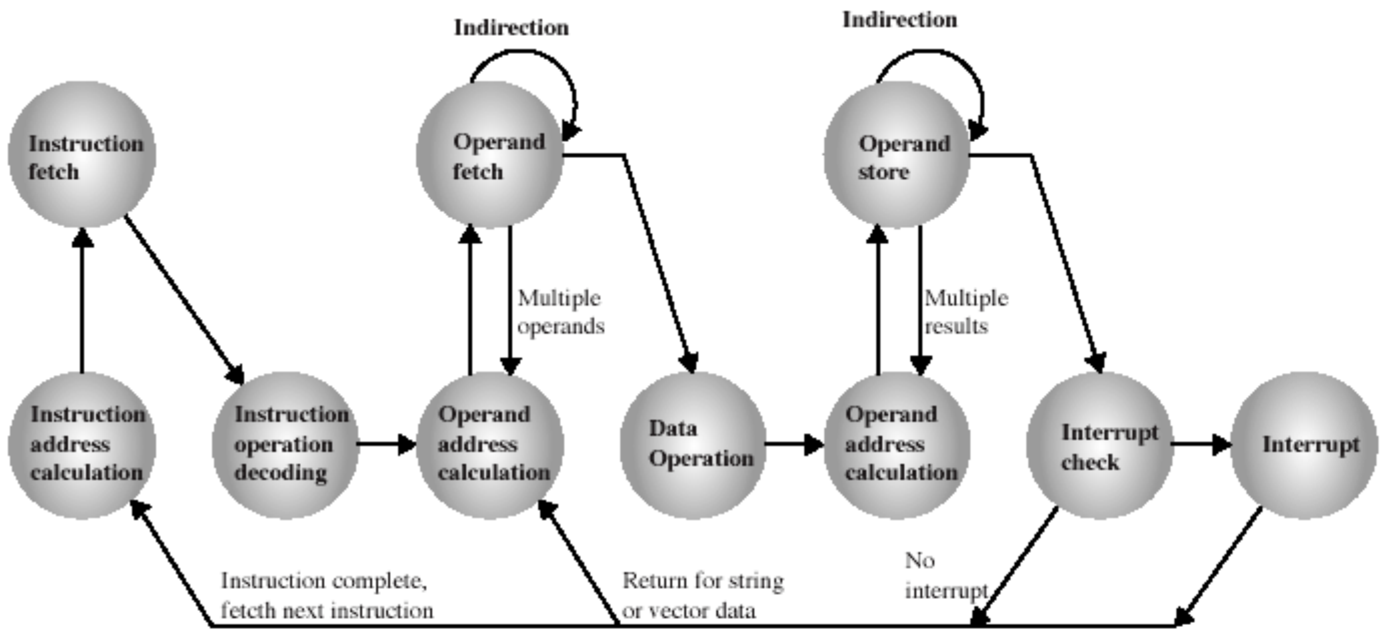

Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction
Return for string or vector data
No interrupt

Figure 12.5 Instruction Cycle State Diagram

Q1: What is the difference in English between Figure 3.12 and Figure 12.5?

Q2: Give an x86 instruction that can be used to discuss Figure 3.12. Using your instruction, show how the instruction goes through the various stages of execution. Also, the following diagram might help.

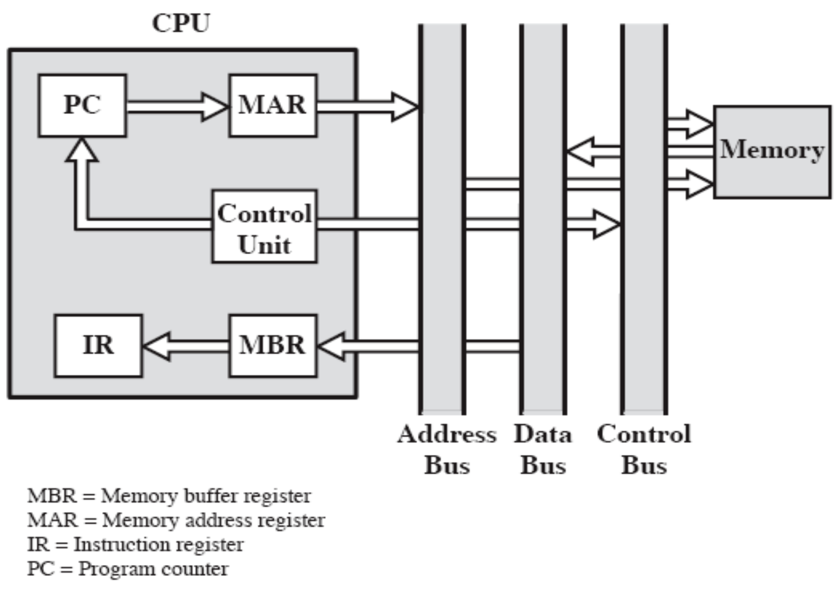

CPU
PC
MAR
Control Unit
IR
MBR
Memory
Address Bus
Data Bus
Control Bus
MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure 12.6 Data Flow, Fetch Cycle

Q3: Do the same thing in Q2 for Figure 12.5. The following diagram might help.

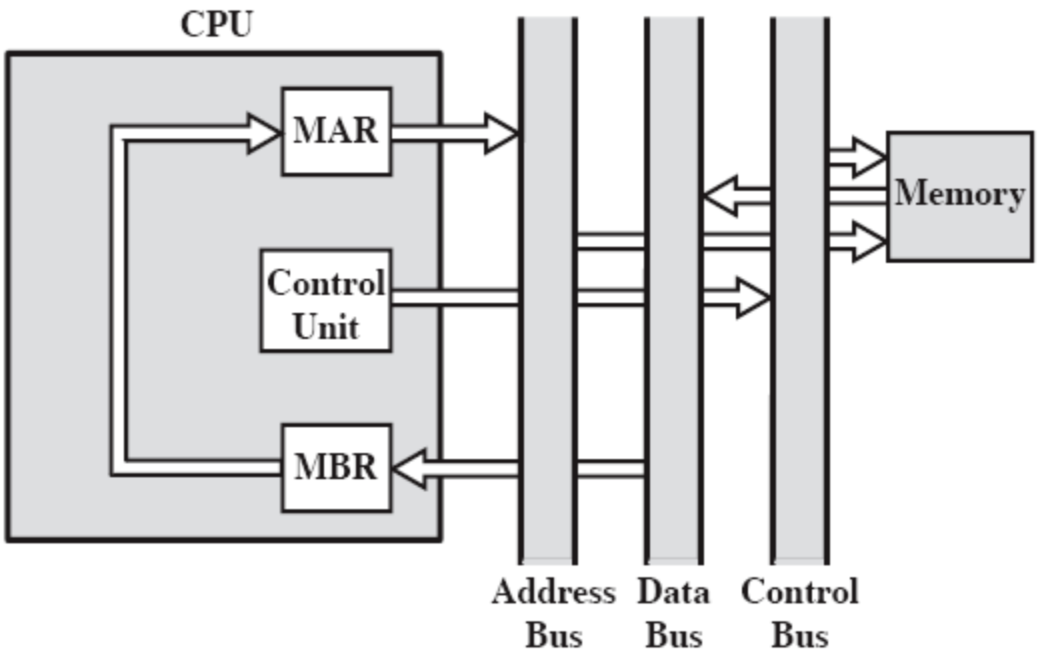


Figure 12.7 Data Flow, Indirect Cycle

Instruction Pipelining

Organization enhancements to the processor over time have greatly improved performance. Examples include multiple general purpose registers instead of a single accumulator and the use of cache and multiple levels of cache to name a few. The next organizational feature to discuss is instruction pipelining.

Pipelining is a technique that improves instruction throughput by executing as many instructions as possible in parallel. In particular, these instructions are in various "stages" of pipeline execution.

Instruction pipelining is very much like the assembly line at a manufacturing plant where products in various stages of production can be worked on simultaneously.

If we examine Figure 12.5 above, we could think of each state as a task; therefore, there are 10 tasks which occur in sequence. Obviously, some kind of instruction pipelining is possible.

Before discussing instruction pipelining in detail, it is useful to discuss the various stages of instruction processing. In particular, we will break up instruction processing into the following six stages:

1. FI (fetch instruction) - get the next instruction
2. DI (decode instruction) - decode the opcode and operands
3. CO (calculate operands) - calculate EA of the operands
4. FO (fetch operands) - fetch operands from memory (not necessary for register data)
5. EI (execute instruction) - execute instruction storing result if necessary
6. WO (write operand) - write the result in MEM

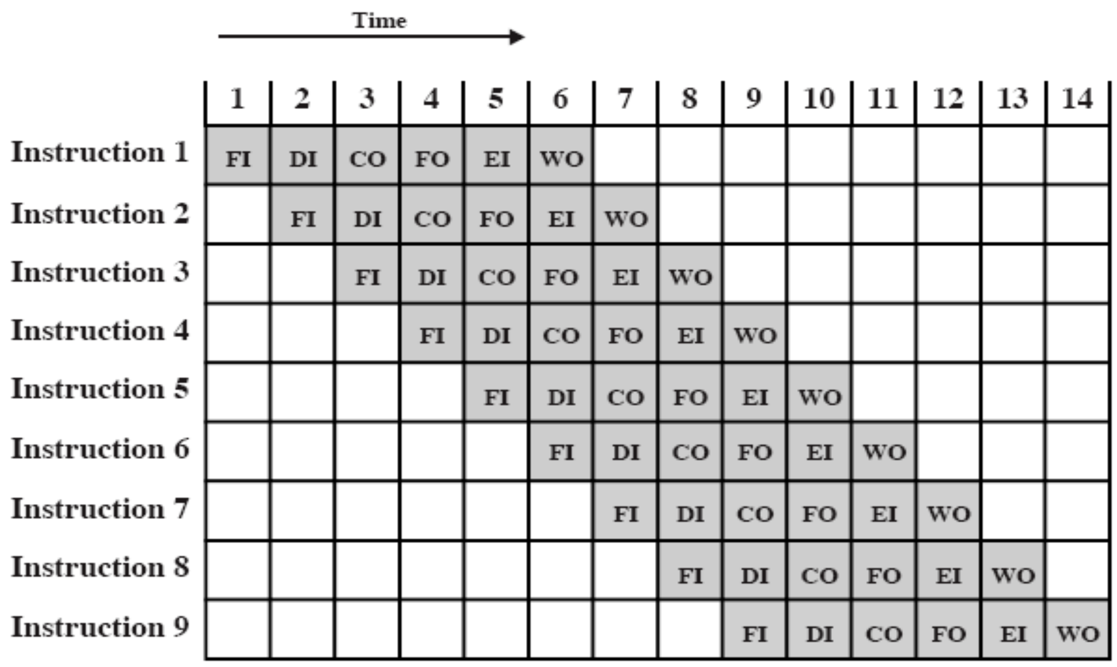
In a perfect world, this would lead us to an architecture that has a CPU with a six stage pipeline where each stage takes an equal duration of time to execute.

Q4: Do the above pipeline stages take the same duration of time to complete their specific operation? If not, give two stages that can have different durations of time to complete their task.

Q5: Assuming no pipelining and each stage in the pipeline takes 1 unit of time to complete, how long would it take 5 instructions to complete?

Q6: Assuming a six stage pipeline and each stage in the pipeline takes 1 unit of time to complete, how long would it take 5 instructions to complete (assuming no delays)?

Answer:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Figure 12.10 Timing Diagram for Instruction Pipeline Operation

Let's examine some quantitative principles of computer design that I will be using for the remainder of the course.

Speedup

"Speedup" is a very simple, useful term that allows us a way to measure performance and execution times.

Speedup = Time(Old)/Time(New)

Example: A trip to downtown Portland takes 50 minutes by car and 30 minutes by MAX. What is the speedup? Answer: speedup=50/30=1.6666

Q7: Car to Seattle takes 3.5 hours and a plane to Seattle takes 1 hour. Compute the speedup.

Q8: A 486 takes 20 minutes to compile a program. A Pentium takes 6 minutes to compile the same program. Compute the speedup.

New is faster than Old if Time(New)<Time(Old)

New is N% faster than Old given: Speedup = Time(Old)/Time(New) = 1+N%

Therefore as an example, speedup of MAX over car is 1.6666; therefore, MAX is 66.66% faster than the car.

Q9: How much faster is the Pentium than the 486?

Notice, it is also the case that:

Speedup = Speed(New)/Speed(Old)

Q10: A plane flies to Seattle at 150MPH(average). I could take a jet that flies at 250MPH. What is the speedup?

Amdahl's Law

Given a job, some fraction of that job may be enhanced.

In particular, if we can make some enhancement to a particular machine that improves performance, speedup will tell us how much faster the machine is with the enhancement versus the machine without the enhancement.

Q11: Let's assume it takes us 1 hour to drive to Portland. From Portland, it takes us 3 hours by train to get to Seattle. What is the speedup if we fly instead and it takes 1 hour to fly to Seattle?

Answer: speedup = Time(Old)/Time(New) = 4/2 = 2.0

Q12: How much faster is the trip if we fly to Seattle versus taking the train?

Answer: Flying is 100% faster than taking the train.

fraction(enhanced) - is the fraction of the "original" time that can be enhanced.

Q13: What is the fraction(enhanced) in Q11?

Answer: 3 hours can be enhanced, so fraction(enhanced) is 3/4=75%.

Further examples to clarify the fraction(enhanced) include the following:

- Speeding up the floating-point calculations doesn't speed up the integer calculations
- If we are traveling to Eugene and speedup the highway driving somehow, that doesn't speedup the packing, walking to the car, ...

An interesting observation is that if we do the fraction(enhanced) at infinite speed, this gives us the maximum speedup for that enhancement.

Back to our story .....

Q14: What is the speedup of the six stage pipelined machine versus the machine with no pipelining?