

CSE140 L

Instructor: Thomas Y. P. Lee

March 8, 2006



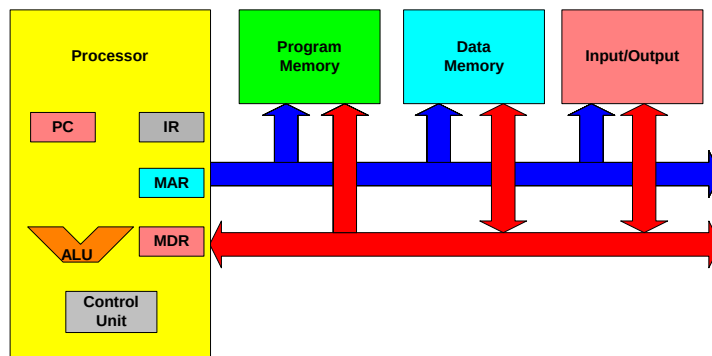
Agenda

- Traditional Computer Architecture
- Instruction Cycles and State Machine
- Methods of Designing a CPU
- Speeding Up the Clock
- Different Styles of Register Transfer
- Different Datapaths without and with Pipeline
- Example of An Adder without and with Pipeline
- Datapath (Hierarchy)
- Datapath (ALU and Registers)
- Instruction Path
- Instruction Fetch Pipeline
- Instruction Decode Pipeline
- ALU Execution Pipeline
- Memory Interface: Harvard vs. Princeton
- Harvard Processor Architecture
- Princeton Processor Architecture
- A Simplified Processor: MIPS 2000
- MIPS Instruction
- RISC CPU Simulation
- Data Hazards and Stalling in Processor
- References



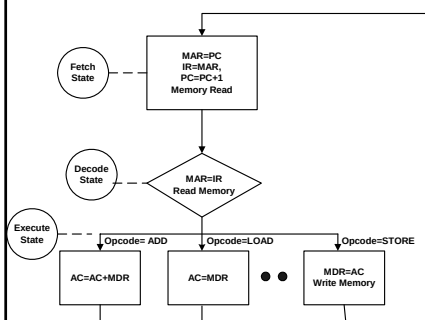
Traditional Computer Architecture

- A traditional computer system consists of three main units: processor (CPU), memory (program, data), I/O
- Complex Instruction Set Computer (CISC): many addressing modes, a number of possible instruction lengths.
- RISC (Reduced Instruction Set Computer): reduce the complexity of the instruction set



Instruction Cycles and State Machine

- Register Transfer Cycle 1: **FETCH** State.
 - $PC = PC + 1$, $MAR = PC$, read memory, $IR = MAR$
- Register Transfer Cycle 2: **DECODE** State
 - Decode Opcode, $MAR = IR$, read memory
- Register Transfer Cycle 2: **EXECUTE** State
 - Execute, $MAR = PC$, and go to **FETCH** state



Instruction Symbol	Opcode	Microoperation
ADD address	00	$AC \leftarrow AC + \text{contents of MAR}$
STORE address	01	Contents of $MAR \leftarrow AC$
LOAD address	02	$AC \leftarrow \text{contents of MAR}$
JUMP address	03	$PC \leftarrow \text{address}$
JNEG address	04	If $AC < 0$ then $PC \leftarrow \text{address}$
SUB address	05	$AC = AC - MDR$
XOR address	06	$AC = AC \text{ XOR } MDR$
OR address	07	$AC = AC \text{ OR } MDR$
AND address	08	$AC = AC \text{ AND } MDR$
JPOS address	09	If $AC > 0$ then $PC \leftarrow \text{address}$
JZERO address	0A	If $AC = 0$ then $PC \leftarrow \text{address}$
ADDI address	0B	$AC = AC + \text{address}$
SHL address	0C	$AC = AC$ shifted left address bits
SHR address	0D	$AC = AC$ shifted right address bits

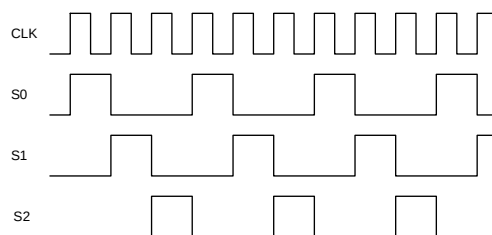
Methods of Designing a CPU

- Analyze instruction set => datapath requirements
 - ❖ the meaning of each instruction is given by the register transfers
 - ❖ datapath must include storage element
 - ❖ datapath must support each register transfer
- Select set of datapath components and establish clocking methodology
- Assemble datapath meeting the requirements
- Analyze implementation of each instruction to determine setting of control points that affects the register transfer.
- Assemble the control logic
- Enhance the design with pipeline and multiple clock cycles
- Advanced subject: enhance the design with instruction cache
- Advanced subject: how to test your CPU?



Speeding Up the Clock

- Performance Index of a machine is determined by:
 - (1) Instruction count
 - (2) Clock cycle time
 - (3) Clock cycles per instruction
 - Single clock cycle per instruction
- Disadvantages: (1) Long cycle time (2) Instruction execute in sequential
- **Why pipeline processing?** the fetch unit fetch instruction N+2, while decode unit decodes instruction n+1, and the execute unit executes instruction n
 - **What is Pipeline Processing? How to design?**
- Pipelining increases instruction throughput by breaking up the instruction into different stages. Each stage is separated by a register which stores the results of the previous stage so that the correct values are used in the next stage. The clock cycle length of the pipelined processor is the time it takes for the slowest stage of the pipeline to complete its function.

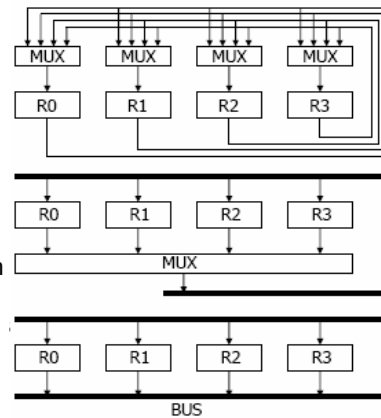


$$\text{CPI} = \frac{\text{Inst. Count}}{\text{Cycle Time}}$$

Different Styles of Registers Transfer



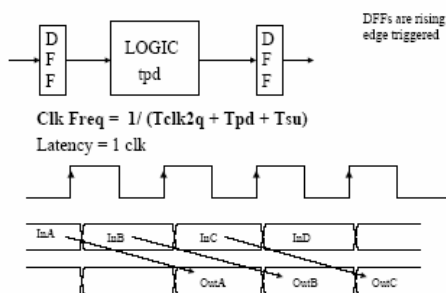
- Point-to-point connection
 - ❖ dedicated wires
 - ❖ muxes on inputs of each register
- Common input from multiplexer
 - ❖ load enables for each register
 - ❖ control signals for multiplexer
- Common bus with output enables
 - ❖ output enables and load enables for each register



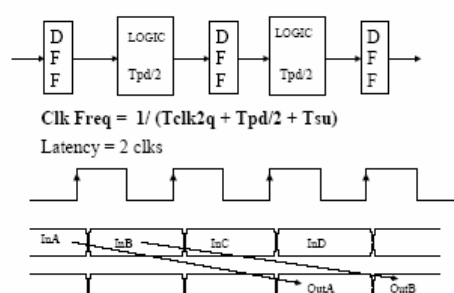
Different Datapaths Without and With Pipeline



Registered Datapath



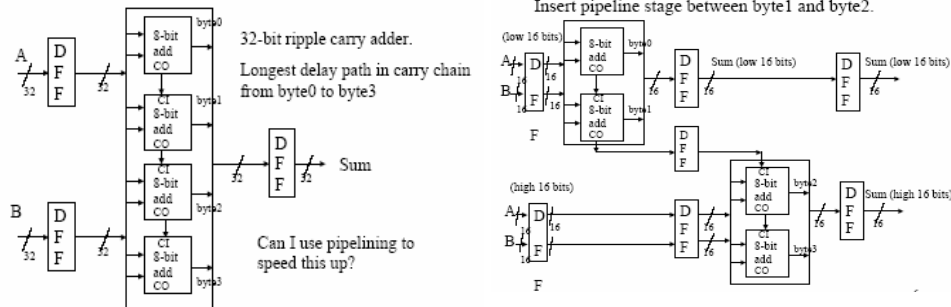
Add a pipeline stage



What is Latency ?

- Number of clock cycles between input value and output value
- Adding pipeline stages always increases latency

Example of An Adder Without and With Pipeline



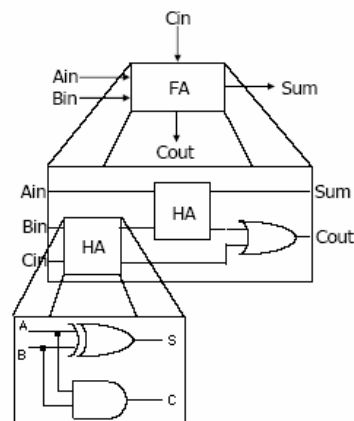
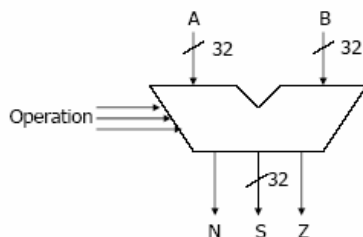
- Note that the pipeline stage broke the carry chain into two equal paths
- ❖ Each pipeline stage should have approximately the same combinational delay
- ❖ Clock speed will be set by the delay of the slowest pipeline stage
- If I inserted 2 pipeline stages, I would need to break the carry chain delay into equal thirds
- Could insert a pipeline stage between each bit in order to get maximum clock speed

Datapath (Hierarchy)



Arithmetic circuits constructed in hierarchical and modular fashion

- each bit in datapath is functionally identical
- 4-bit, 8-bit, 16-bit, 32-bit datapaths



Datpath (ALU and Registers)



Accumulator

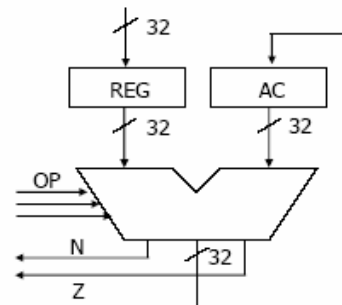
- special register
- one of the inputs to ALU
- output of ALU stored back in accumulator

One-address instructions

- operation and address of one operand
- other operand and destination is accumulator register
- $AC \leftarrow AC \text{ op Mem}[\text{addr}]$
- "single address instructions" (AC implicit operand)

Multiple registers

- part of instruction used to choose register operands



Instruction Path



▪ Program counter (PC)

- ❖ keeps track of program execution
- ❖ address of next instruction to read from memory
- ❖ may have auto-increment feature or use ALU

▪ Instruction register (IR)

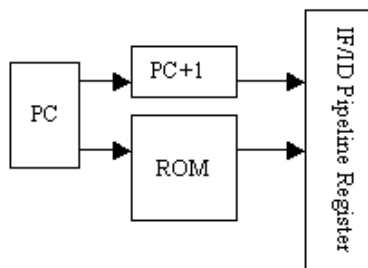
- ❖ current instruction
- ❖ includes ALU operation and address of operand
- ❖ also holds target of jump instruction
- ❖ immediate operands

▪ Relationship to data path

- ❖ PC may be incremented through ALU
- ❖ contents of IR may also be required as input to ALU –immediate operands

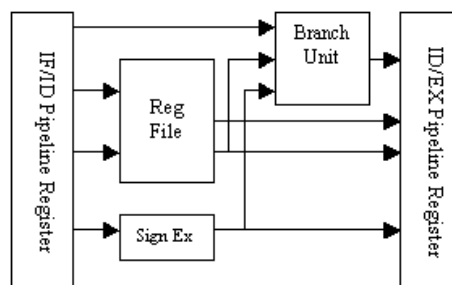
Instruction Fetch Pipeline

- The Instruction Fetch (IF) stage is responsible for obtaining the requested instruction from memory.
- The instruction and the program counter (which is incremented to the next instruction) are stored in the IF/ID pipeline register as temporary storage so that may be used in the next stage at the start of the next clock cycle.



Instruction Decode Pipeline

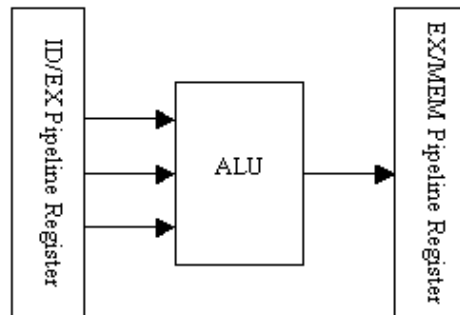
- The Instruction Decode (ID) stage is responsible for decoding the instruction and sending out the various control lines to the other parts of the processor.
- The instruction is sent to the control unit where it is decoded and the read registers are fetched from the register file. The ID stage is also where branching occurs.



ALU Execution Pipeline



- The Execution (EX) stage is where any calculations are performed. The main component in this stage is the ALU. The ALU is made up of arithmetic, logic and shifting capabilities.



Datapath of Memory Interface – Harvard vs. Princeton Architecture



- Memory
 - ❖ separate data and instruction memory (Harvard architecture)
 - Two address busses, two data busses
 - ❖ Single combined memory (Princeton architecture)
 - single address bus, single data bus

Separate memory

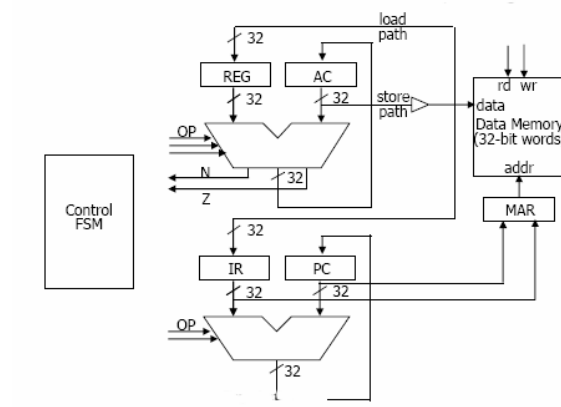
- ❖ ALU output goes to data memory input
- ❖ register input from data memory output
- ❖ data memory address from instruction register
- ❖ instruction register from instruction memory output
- ❖ instruction memory address from program counter

Single memory

- ❖ address from PC or IR
- ❖ memory output to instruction and data registers
- ❖ memory input from ALU output

-

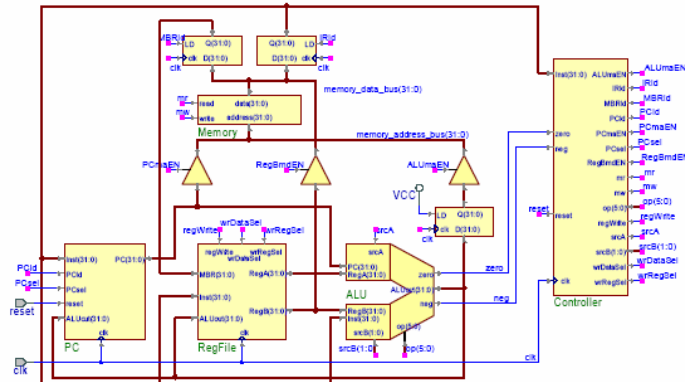
- ❖ which register outputs are connected to which register inputs
- ❖ arrows represent data-flow, other are control signals from control FSM
- ❖ MAR may be a simple multiplexer rather than separate register (impl. using 3-state)
- ❖ two MBRs (REG and IR)
- ❖ load control for each register



A Simplified Princeton Processor – MIPS R2000

Modeled after MIPS R2000

- used as main example in 378 textbook by Patterson & Hennessy
- Princeton architecture – shared data/instruction memory
- 32-bit machine
- 32 register file
- PC incremented through ALU
- Multi-cycle instructions in our implementation, single-cycle for real R2000
- Only a subset of the instructions are implemented
- Synchronous Mealy or Moore controller



The MIPS Instruction

- All MIPS instructions are 32 bits long. The three instruction formats:

	31	26	21	16	11	6	0
❖ R-type	op	rs	rt	rd	shamt	funct	
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
❖ I-type	31	26	21	16			0
	op	rs	rt	immediate			
	6 bits	5 bits	5 bits	16 bits			
❖ J-type	31	26					0
	op	target address					
	6 bits	26 bits					

- The different fields are:
 - ❖ op: operation of the instruction
 - ❖ rs, rt, rd: the source and destination register specifiers
 - ❖ shamt: shift amount
 - ❖ funct: selects the variant of the operation in the “op” field
 - ❖ address / immediate: address offset or immediate value
 - ❖ target address: target address of the jump instruction

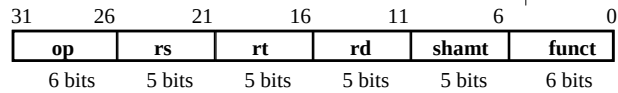
MIPS Instruction (Cont.)



- ADD and SUB

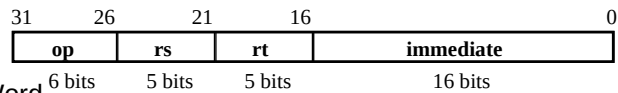
- ❖ addU rd, rs, rt

- ❖ subU rd, rs, rt



- OR Immediate:

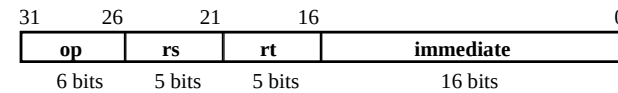
- ❖ ori rt, rs, imm16



- LOAD and STORE Word

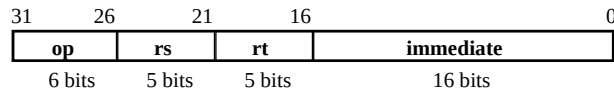
- ❖ lw rt, rs, imm16

- ❖ sw rt, rs, imm16



- BRANCH:

- ❖ beq rs, rt, imm16



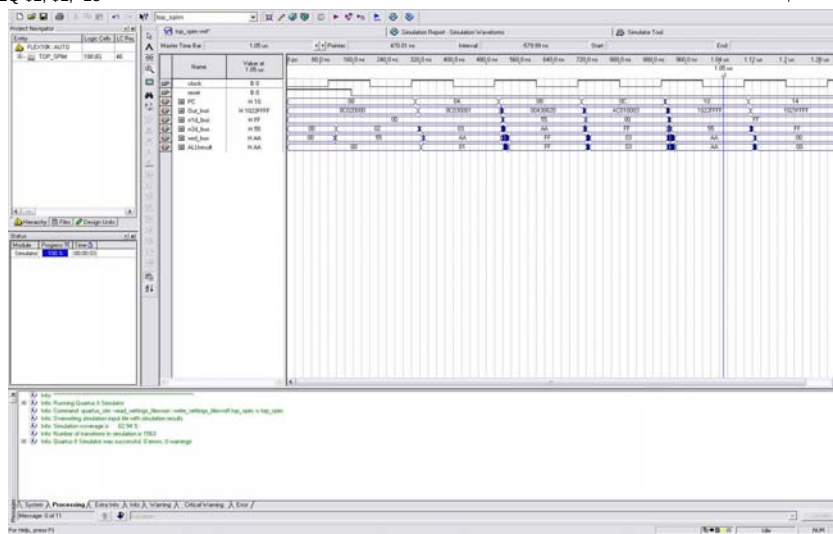
For detail on MIPS CPU design, see References, use the source code Provided, try to compile on your own QuartusII, and run simulation!

The program is:
LW \$2, 00
LW \$3, 01
ADD \$1, \$2, \$3
SW \$1, 03
BEQ \$1, \$2, -4
BEQ \$1, \$1, -18

RISC CPU Simulation

- ❖ Please see class web site for VHDL source code, done by TA !! Reference #3

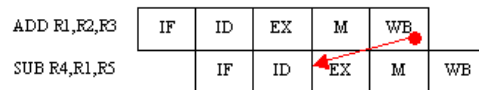
- ❖ Can you change the code to include pipeline processing?



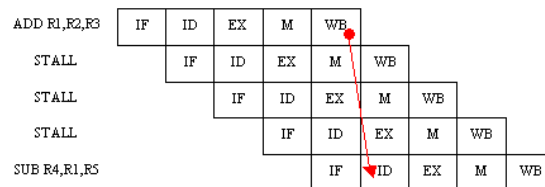
Data Hazards and Stalling in Processor



- Data Hazards happens when an instruction attempts to use a register whose value depends on the result of previous instructions that have not yet finished.



- Stalling involves halting the flow of instructions until the required result is ready to be used. The simplest way to resolve a data hazard. However, stalling wastes processor time by doing nothing while waiting for the result.



References



1. M. Morris Mano: Logic and Computer Design Fundamentals, 3rd edition
2. Glen G. Langdon, Computer Design
3. Hamblen/Furman: Rapid Prototyping of Digital Systems

The following material relate to MIPS CPU design

3. <http://www.cs.washington.edu/education/courses/cse370>
4. Our class web site contains an example of VHDL code of MIPS CPU

The following material relate to RISC CPU design

5. <http://murray.newcastle.edu.au/users/students/1999/c9311421/index.html>
 6. <http://www.opencores.com/projects.cgi/web/riscmcu/overview>
RISCMCU is based on the features and instruction set of Atmel AVR AT90S1200 RISC Microcontroller, include VHDL source code, thesis (design principle), presentation
 7. Jan Gray, "Building a RISC System In an FPGA" Part I, II and III, March 2000, Circuit Cellar Magazine
- Notice that all examples DID not include the pipeline design

Important Notices



1. 3/15/06, 10AM @ B270 , TA will have oral exam with every group, please Prepare your lab report for oral presentation
2. 3/21/06, 9:00AM ~ 10:20 AM CSE140L final exam