

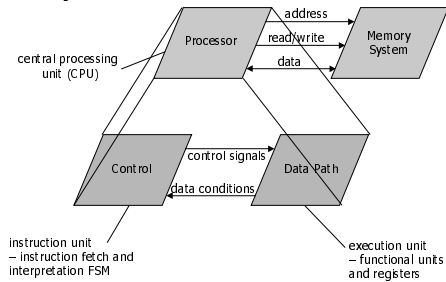
Computer organization

- Computer design – an application of digital logic design procedures
- Computer = processing unit + memory system
- Processing unit = control + datapath
- Control = finite state machine
 - inputs = machine instruction, datapath conditions
 - outputs = register transfer control signals, ALU operation codes
 - instruction interpretation = instruction fetch, decode, execute
- Datapath = functional units + registers
 - functional units = ALU, multipliers, dividers, etc.
 - registers = program counter, shifters, storage registers

CSE 370 - Winter 2000 - Computer Organization - 1

Structure of a computer

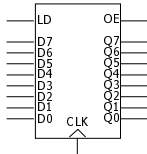
- Block diagram view



CSE 370 - Winter 2000 - Computer Organization - 2

Registers

- Selectively loaded – EN or LD input
- Output enable – OE input
- Multiple registers – group 4 or 8 in parallel



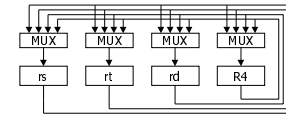
OE asserted causes FF state to be connected to output pins; otherwise they are left unconnected (high impedance)

LD asserted during a lo-to-hi clock transition loads new data into FFs

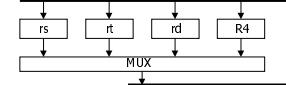
CSE 370 - Winter 2000 - Computer Organization - 3

Register transfer

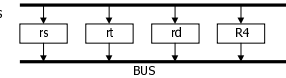
- Point-to-point connection
 - dedicated wires
 - muxes on inputs of each register



- Common input from multiplexer
 - load enables for each register
 - control signals for multiplexer



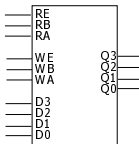
- Common bus with output enables
 - output enables and load enables for each register



CSE 370 - Winter 2000 - Computer Organization - 4

Register files

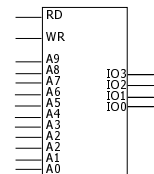
- Collections of registers in one package
 - two-dimensional array of FFs
 - address used as index to a particular word
 - can have separate read and write addresses so can do both at same time
- 4 by 4 register file
 - 16 D-FFs
 - organized as four words of four bits each
 - write-enable (load)
 - read-enable (output enable)



CSE 370 - Winter 2000 - Computer Organization - 5

Memories

- Larger collections of storage elements
 - implemented not as FFs but as much more efficient latches
 - high-density memories use 1 to 5 switches (transistors) per memory bit
- Static RAM – 1024 words each 4 bits wide
 - once written, memory holds forever (not true for denser dynamic RAM)
 - address lines to select word (10 lines for 1024 words)
 - read enable
 - same as output enable
 - often called chip select
 - permits connection of many chips into larger array
 - write enable (same as load enable)
 - bi-directional data lines
 - output when reading, input when writing



CSE 370 - Winter 2000 - Computer Organization - 6

Instruction sequencing

- Example – an instruction to add the contents of two registers (Rx and Ry) and place result in a third register (Rz)
- Step 1: get the ADD instruction from memory into an instruction register
- Step 2: decode instruction
 - instruction in IR has the code of an ADD instruction
 - register indices used to generate output enables for registers Rx and Ry
 - register index used to generate load signal for register Rz
- Step 3: execute instruction
 - enable Rx and Ry output and direct to ALU
 - setup ALU to perform ADD operation
 - direct result to Rz so that it can be loaded into register

Instruction types

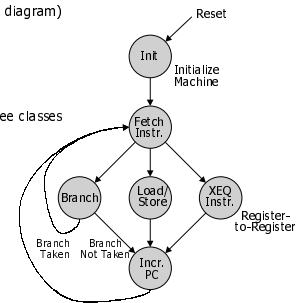
- Data manipulation
 - add, subtract
 - increment, decrement
 - multiply
 - shift, rotate
 - immediate operands
- Data staging
 - load/store data to/from memory
 - register-to-register move
- Control
 - conditional/unconditional branches in program flow
 - subroutine call and return

Elements of the control unit (aka instruction unit)

- Standard FSM elements
 - state register
 - next-state logic
 - output logic (datapath/control signalling)
 - Moore or synchronous Mealy machine to avoid loops unbroken by FF
- Plus additional "control" registers
 - instruction register (IR)
 - program counter (PC)
- Inputs/outputs
 - outputs control elements of data path
 - inputs from data path used to alter flow of program (test if zero)

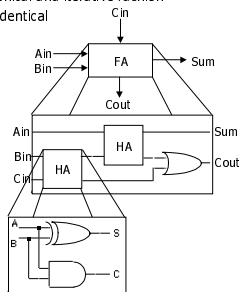
Instruction execution

- Control state diagram (for each diagram)
 - reset
 - fetch instruction
 - decode
 - execute
- Instructions partitioned into three classes
 - branch
 - load/store
 - register-to-register
- Different sequence through diagram for each instruction type



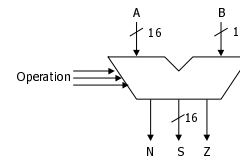
Data path (heirarchy)

- Arithmetic circuits constructed in hierarchical and iterative fashion
 - each bit in datapath is functionally identical
 - 4-bit, 8-bit, 16-bit, 32-bit datapaths



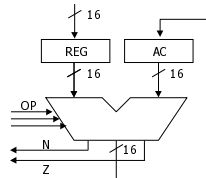
Data path (ALU)

- ALU block diagram
 - input: data and operation to perform
 - output: result of operation and status information



Data path (ALU + registers)

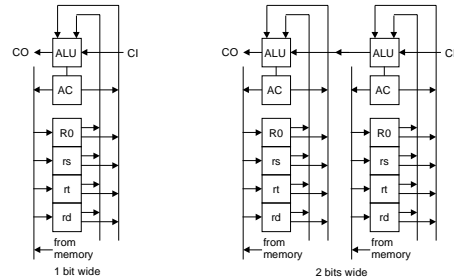
- Accumulator
 - special register
 - one of the inputs to ALU
 - output of ALU stored back in accumulator
- One-address instructions
 - operation and address of one operand is accumulator register
 - other operand and destination is accumulator register
 - AC ← AC op Mem[addr]
 - "single address instructions" (AC implicit operand)
- Multiple registers
 - part of instruction used to choose register operands



CSE 370 - Winter 2000 - Computer Organization - 13

Data path (bit-slice)

- Bit-slice concept – iterate to build n-bit wide datapaths



CSE 370 - Winter 2000 - Computer Organization - 14

Instruction path

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

CSE 370 - Winter 2000 - Computer Organization - 15

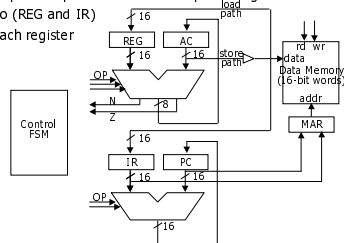
Data path (memory interface)

- Memory
 - separate data and instruction memory (Harvard architecture)
 - two address buses, two data buses
 - 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

CSE 370 - Winter 2000 - Computer Organization - 16

Block diagram of processor

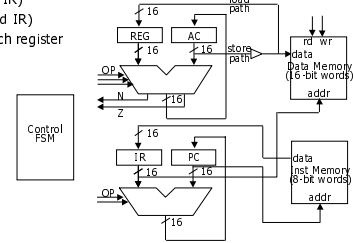
- Register transfer view of Princeton architecture
 - which register outputs are connected to which register inputs
 - arrows represent data-flow, other are control signals from control FSM
 - MAR may be a simple multiplexer rather than separate register
 - MBR is split in two (REG and IR)
 - load control for each register



CSE 370 - Winter 2000 - Computer Organization - 17

Block diagram of processor

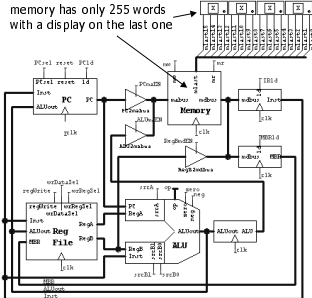
- Register transfer view of Harvard architecture
 - which register outputs are connected to which register inputs
 - arrows represent data-flow, other are control signals from control FSM
 - two MARs (PC and IR)
 - two MBRs (REG and IR)
 - load control for each register



CSE 370 - Winter 2000 - Computer Organization - 18

CSE370 processor data-path and memory

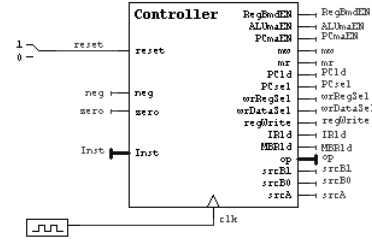
- Princeton architecture
- Register file
- Instruction register
- PC incremented through ALU
- Modeled after MIPS r000 (used in 378 textbook by Patterson & Hennessy)
 - really a 32 bit machine
 - we'll do a 16 bit version



CSE 370 - Winter 2000 - Computer Organization - 19

CSE370 processor control

- Synchronous Mealy machine
- Multiple cycles per instruction



CSE 370 - Winter 2000 - Computer Organization - 20

CSE370 processor instructions

- Three principal types (16 bits in each instruction)

type	op	rs	rt	rd	funct
R(register)	3	3	3	3	4
I(immediate)	3	3	3	7	
J(jump)	3	13			

- Instructions we'll implement

	op	rs	rt	rd	funct	description	
R	add	0	rs	rt	rd	0	rd = rs + rt
	sub	0	rs	rt	rd	1	rd = rs - rt
	and	0	rs	rt	rd	2	rd = rs & rt
	or	0	rs	rt	rd	3	rd = rs rt
	slt	0	rs	rt	rd	4	rd = (rs < rt)
I	lw	1	rs	rt	offset		rt = mem[rs + offset]
	sw	2	rs	rt	offset		mem[rs + offset] = rt
	beq	3	rs	rt	offset		pc = pc + offset, if (rs == rt)
	addi	4	rs	rt	offset		rt = rs + offset
J	j	5	target address				pc = target address
	halt	7	-				stop execution until reset

CSE 370 - Winter 2000 - Computer Organization - 21

Tracing an instruction's execution

- Instruction: r3 = r1 + r2

R	op	rs	rt	rd	funct
0		r1	r2	r3	0

- instruction fetch
 - move instruction address from PC to memory address bus
 - assert memory read
 - move data from memory data bus into IR
 - configure PC to store new value from ALUout
- instruction decode
 - op-code bits of IR are input to control FSM
 - rest of IR bits encode the operand addresses (rs and rt)
 - these go to register file

CSE 370 - Winter 2000 - Computer Organization - 22

Tracing an instruction's execution (cont'd)

- Instruction: r3 = r1 + r2

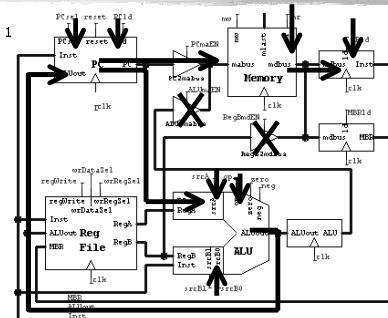
R	op	rs	rt	rd	funct
0		r1	r2	r3	0

- instruction execute
 - set up ALU inputs
 - configure ALU to perform ADD operation
 - configure register file to store ALU result (rd)

CSE 370 - Winter 2000 - Computer Organization - 23

Tracing an instruction's execution (cont'd)

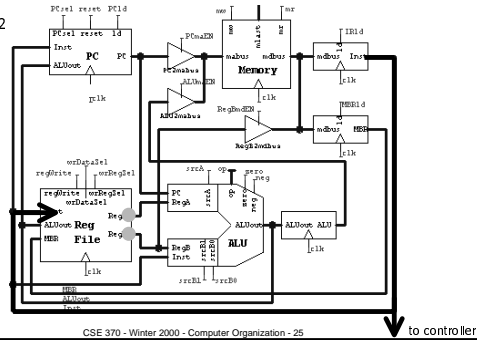
- Step 1



CSE 370 - Winter 2000 - Computer Organization - 24

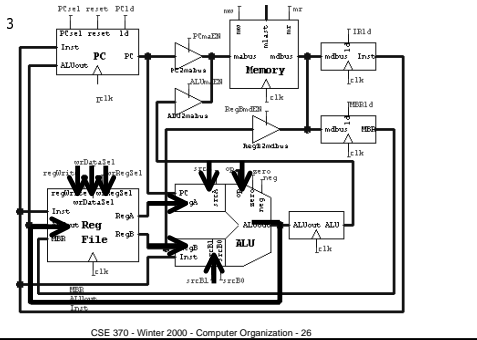
Tracing an instruction's execution (cont'd)

Step 2



Tracing an instruction's execution (cont'd)

Step 3



Register-transfer-level description

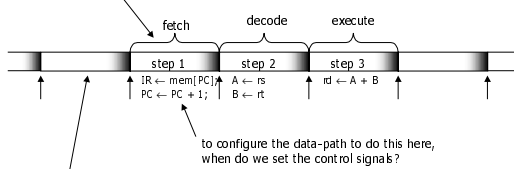
- Control
 - transfer data between registers by asserting appropriate control signals
- Register transfer notation - work from register to register
 - instruction fetch:
 - $mabus \leftarrow PC$; - move PC to memory address bus (PCmaEN, ALUmaEN)
 - memory read; - assert memory read signal (mr, RegBmdEN)
 - $IR \leftarrow memory$; - load IR from memory data bus (IRld)
 - $op \leftarrow add$ - send PC into A input, 1 into B input, add
 - $PC \leftarrow ALUout$ - load result of incrementing in ALU into PC (PCld, PCset)
 - instruction decode:
 - IR to controller
 - values of A and B read from register file (rs, rt)
 - instruction execution:
 - $op \leftarrow add$ - send regA into A input, regB into B input, add
 - $(srcA, srcB, srcB1, op)$
 - $rd \leftarrow ALUout$ - store result of add into destination register
 - $(regWrite, wrDataSel, wRegSel)$

Register-transfer-level description (cont'd)

- How many states are needed to accomplish these transfers?
 - data dependencies (where do values that are needed come from?)
 - resource conflicts (ALU, busses, etc.)
- In our case, it takes three cycles
 - one for each step
 - all operation within a cycle occur between rising edges of the clock
- How do we set all of the control signals to be output by the state machine?
 - depends on the type of machine (Mealy, Moore, synchronous Mealy)

Review of FSM timing

here if its a Moore machine or asynchronous Mealy machine, e.g., the value of IRld is derived from the value of state bits, in this case, "fetch"

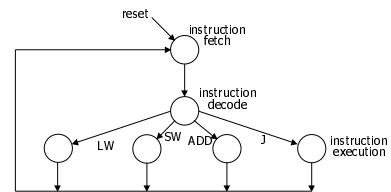


to configure the data-path to do this here, when do we set the control signals?

here if its a synchronous Mealy machine, e.g., IRld = 1 in this state will not change the value of IRld until the next clock edge

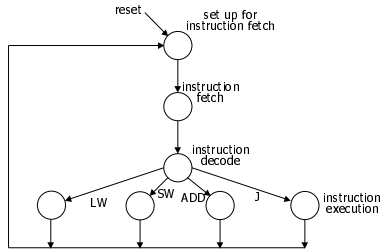
FSM controller for CPU (skeletal Moore FSM)

- First pass at deriving the state diagram (Moore or Mealy machine)
 - these will be further refined into sub-states



FSM controller for CPU (skeletal sync. Mealy FSM)

- First pass at deriving the state diagram (synchronous Mealy machine)
 - these will be further refined into sub-states

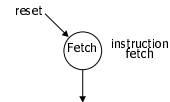


CSE 370 - Winter 2000 - Computer Organization - 31

FSM controller for CPU (reset and inst. fetch)

- Assume Moore machine
 - outputs associated with states rather than arcs
- Reset state and instruction fetch sequence
 - On reset (go to Fetch state)
 - start fetching instructions
 - PC will set itself to zero

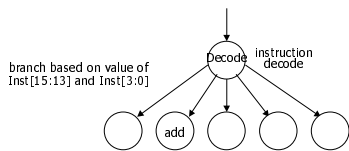
$mabus \leftarrow PC;$
 $memory\ read;$
 $IR \leftarrow memory\ data\ bus;$
 $PC \leftarrow PC + 1;$



CSE 370 - Winter 2000 - Computer Organization - 32

FSM controller for CPU (decode)

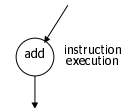
- Operation decode state
 - next state branch based on operation code in instruction
 - read two operands out of register file
 - what if the instruction doesn't have two operands?



CSE 370 - Winter 2000 - Computer Organization - 33

FSM controller for CPU (instruction execution)

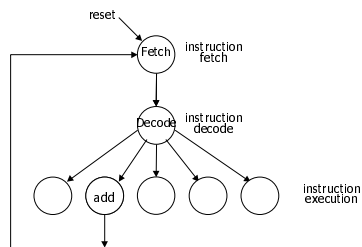
- For add instruction
 - configure ALU and store result in register
 - $rd \leftarrow A + B$
 - other instructions may require multiple cycles



CSE 370 - Winter 2000 - Computer Organization - 34

FSM controller for CPU (add instruction)

- Putting it all together and closing the loop
 - the famous instruction fetch decode execute cycle



CSE 370 - Winter 2000 - Computer Organization - 35

FSM controller for CPU

- Now we need to repeat this for all the instructions of our processor
 - fetch and decode states stay the same
 - different execution states for each instruction
 - some may require multiple states

CSE 370 - Winter 2000 - Computer Organization - 36