
Chapter 3 General-Purpose Processors: Software

ICE!

- Team Problems:
 - Using the simple instruction set, count the occurrences of zero in an array stored in memory locations $M[100]$ through $M[199]$.

A Simple (Trivial) Instruction Set

Assembly instruct.	First byte	Second byte	Operation
MOV Rn, direct	0000 Rn	direct	$Rn = M(\text{direct})$
MOV direct, Rn	0001 Rn	direct	$M(\text{direct}) = Rn$
MOV @Rn, Rm	0010 Rn	Rm	$M(Rn) = Rm$
MOV Rn, #immed.	0011 Rn	immediate	$Rn = \text{immediate}$
ADD Rn, Rm	0100 Rn	Rm	$Rn = Rn + Rm$
SUB Rn, Rm	0101 Rn	Rm	$Rn = Rn - Rm$
JZ Rn, immediate	0110 Rn	immediate	PC = immediate (only if Rn is 0)

⏟
opcode
⏟
operands

Programmer Considerations

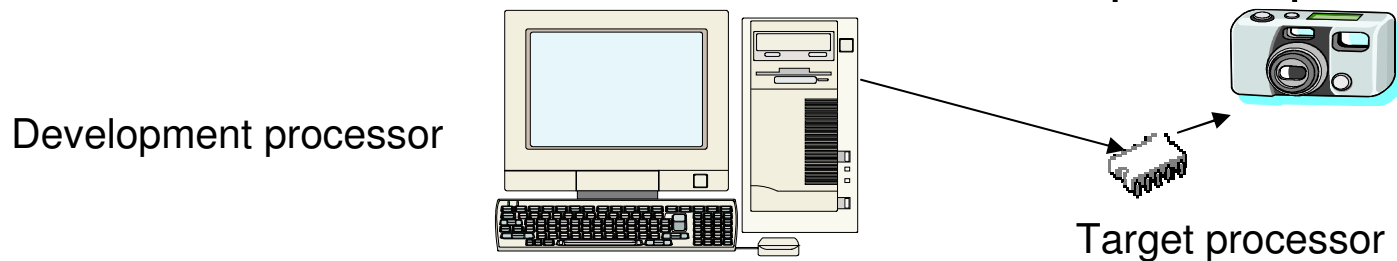
- Program and data memory space
 - Embedded processors often very limited
 - e.g., 64 Kbytes program, 256 bytes of RAM (expandable)
- Registers: How many are there?
 - Only a direct concern for assembly-level programmers
- I/O
 - How communicate with external signals?
- Interrupts

Operating System

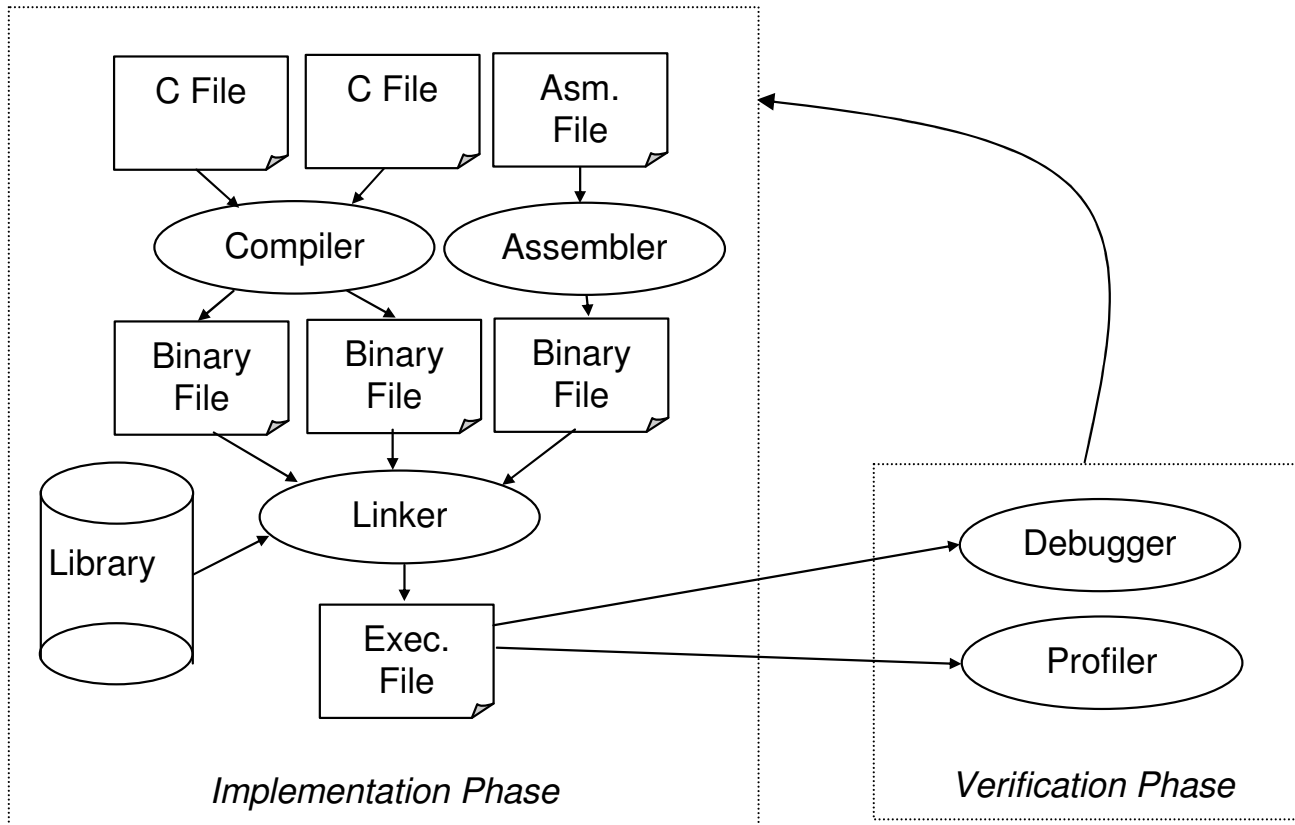
- Optional software layer providing low-level services to a program (application).
 - File management, disk access
 - Keyboard/display interfacing
 - Scheduling multiple programs for execution
- Program makes system calls to the OS

Development Environment

- Development processor
 - The processor on which we write and debug our programs
 - Usually a PC
- *Target processor*
 - The processor that the program will run on in our embedded system
 - Often different from the development processor



Software Development Process

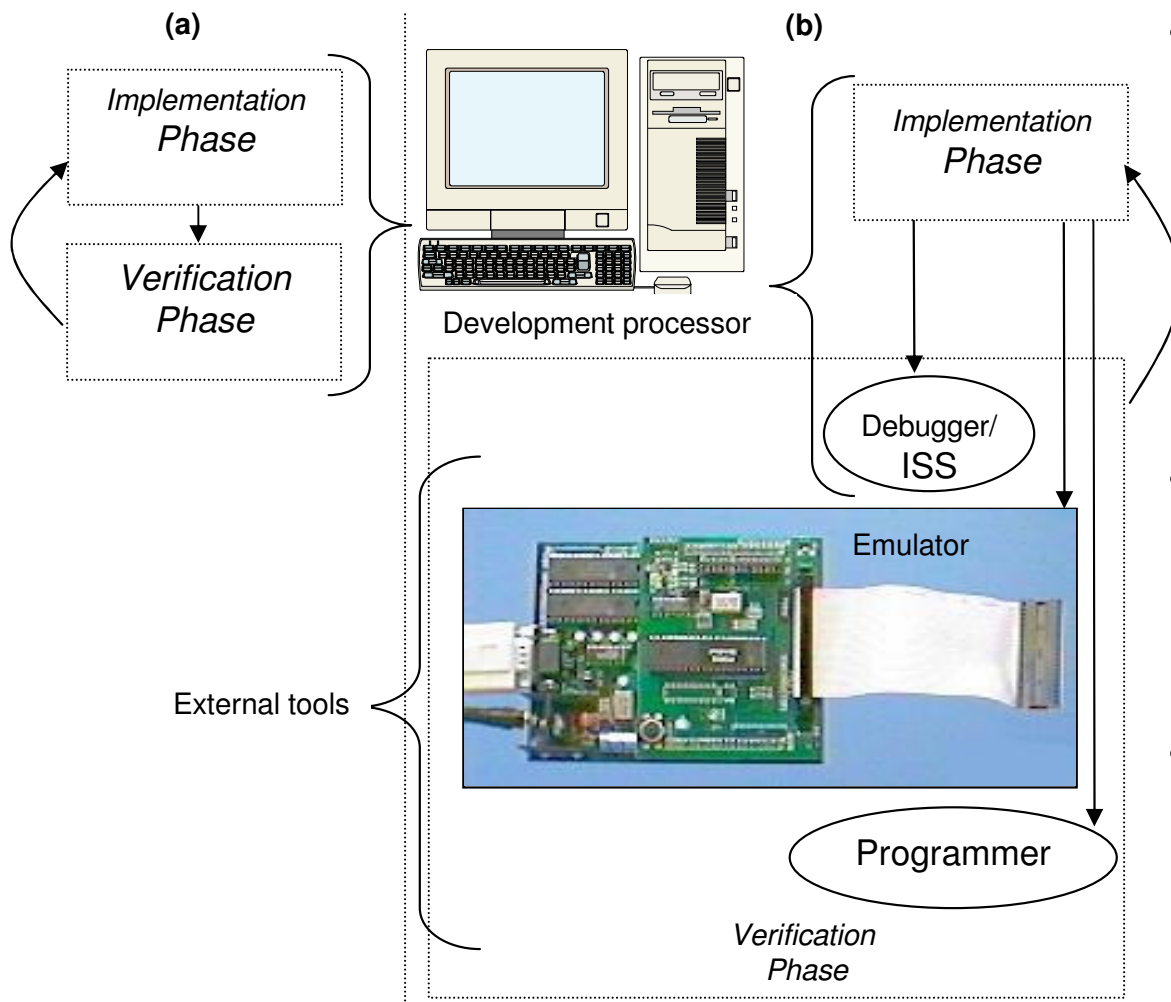


- Compilers
 - Cross compiler
 - Runs on one processor, but generates code for another
- Assemblers
- Linkers
- Debuggers
- Profilers

Running a Program

- If development processor is different than target, how can we run our compiled code? Two options:
 - Download to target processor
 - Simulate
- Simulation
 - One method: Hardware description language
 - But slow, not always available
 - Another method: *Instruction set simulator (ISS)*
 - Runs on development processor, but executes instructions of target processor

Testing and Debugging



- ISS
 - Gives us control over time
 - set breakpoints, look at register values, set values, step-by-step execution, ...
 - But, doesn't interact with real environment
- Download to board
 - Use device programmer
 - Runs in real environment, but not controllable
- Compromise: emulator
 - Runs in real environment, at speed or near
 - Supports some controllability from the PC

Application-Specific Instruction-Set Processors (ASIPs)

- General-purpose processors
 - Sometimes too general to be effective in demanding application
 - e.g., video processing – requires huge video buffers and operations on large arrays of data, inefficient on a GPP
 - But single-purpose processor has high NRE, not programmable
- ASIPs – targeted to a particular domain
 - Contain architectural features specific to that domain
 - e.g., embedded control, digital signal processing, video processing, network processing, telecommunications, etc.
 - Still programmable

Another Common ASIP: Digital Signal Processors (DSP)

- For signal processing applications
 - Large amounts of digitized data, often streaming
 - Data transformations must be applied fast
 - e.g., cell-phone voice filter, digital TV, music synthesizer
- DSP features
 - Several instruction execution units
 - Multiple-accumulate single-cycle instruction, other instrs.
 - Efficient vector operations – e.g., add two arrays
 - Vector ALUs, loop buffers, etc.

Trend: Even More Customized ASIPs

- In the past, microprocessors were acquired as chips
- Today, we increasingly acquire a processor as Intellectual Property (IP)
 - e.g., synthesizable VHDL model
- Opportunity to add a custom datapath hardware and a few custom instructions, or delete a few instructions
 - Can have significant performance, power and size impacts
 - Problem: need compiler/debugger for customized ASIP
 - Remember, most development uses structured languages
 - One solution: automatic compiler/debugger generation
 - e.g., www.tensillica.com

Selecting a Microprocessor

- Issues
 - Technical: speed, power, size, cost
 - Other: development environment, prior expertise, licensing, etc.
- Speed: how evaluate a processor's speed?
 - Clock speed – but instructions per cycle may differ
 - Instructions per second – but work per instr. may differ
 - Dhrystone: Synthetic benchmark, developed in 1984. Dhrystones/sec.
 - MIPS: 1 MIPS = 1757 Dhrystones per second (based on Digital's VAX 11/780). A.k.a. Dhrystone MIPS. Commonly used today.
 - So, 750 MIPS = $750 \times 1757 = 1,317,750$ Dhrystones per second
 - SPEC: set of more realistic benchmarks, but oriented to desktops
 - EEMBC – EDN Embedded Benchmark Consortium, www.eembc.org
 - Suites of benchmarks: automotive, consumer electronics, networking, office automation, telecommunications

General Purpose Processors

Processor	Clock speed	Periph.	Bus Width	MIPS	Power	Trans.	Price
General Purpose Processors							
Intel PIII	1GHz	2x16 K L1, 256K L2, MMX	32	~900	97W	~7M	\$900
IBM PowerPC 750X	550 MHz	2x32 K L1, 256K L2	32/64	~1300	5W	~7M	\$900
MIPS R5000	250 MHz	2x32 K 2 way set assoc.	32/64	NA	NA	3.6M	NA
StrongARM SA-110	233 MHz	None	32	268	1W	2.1M	NA
Microcontroller							
Intel 8051	12 MHz	4K ROM, 128 RAM, 32 I/O, Timer, UART	8	~1	~0.2W	~10K	\$7
Motorola 68HC811	3 MHz	4K ROM, 192 RAM, 32 I/O, Timer, WDT, SPI	8	~.5	~0.1W	~10K	\$5
Digital Signal Processors							
TI C5416	160 MHz	128K, SRAM, 3 T1 Ports, DMA, 13 ADC, 9 DAC	16/32	~600	NA	NA	\$34
Lucent DSP32C	80 MHz	16K Inst., 2K Data, Serial Ports, DMA	32	40	NA	NA	\$75

Sources: Intel, Motorola, MIPS, ARM, TI, and IBM Website/Datasheet; Embedded Systems Programming, Nov. **1998**

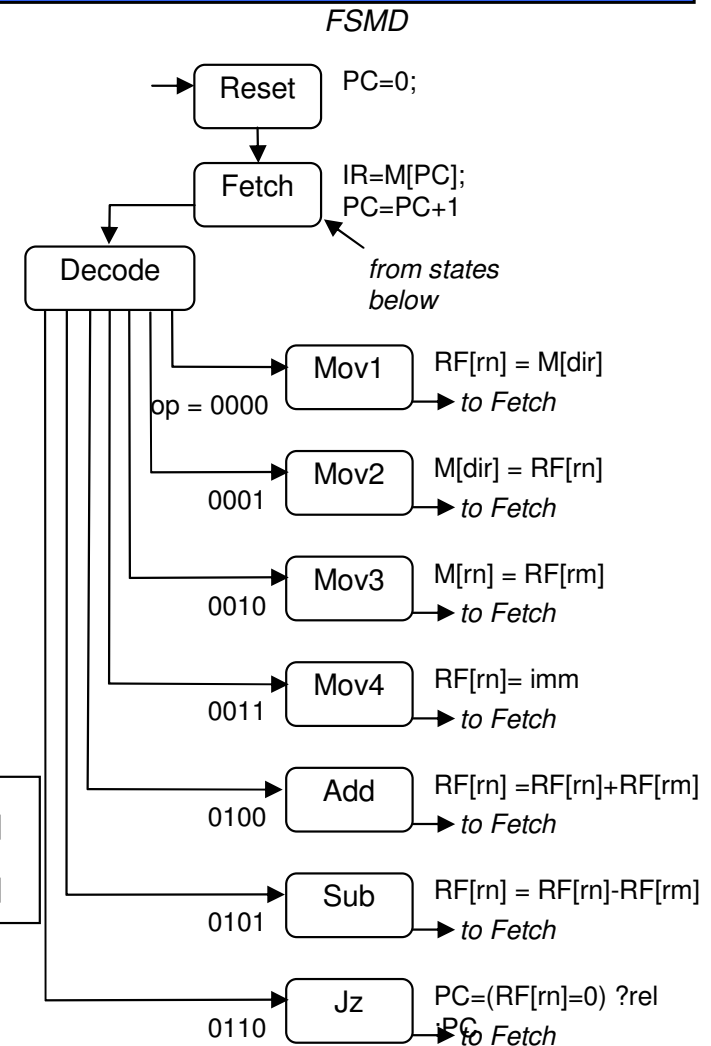
Designing a General Purpose Processor

- Not something an embedded system designer normally would do
 - But instructive to see how simply we can build one top down
 - Remember that real processors aren't usually built this way

- Much more optimized, much more bottom-up design

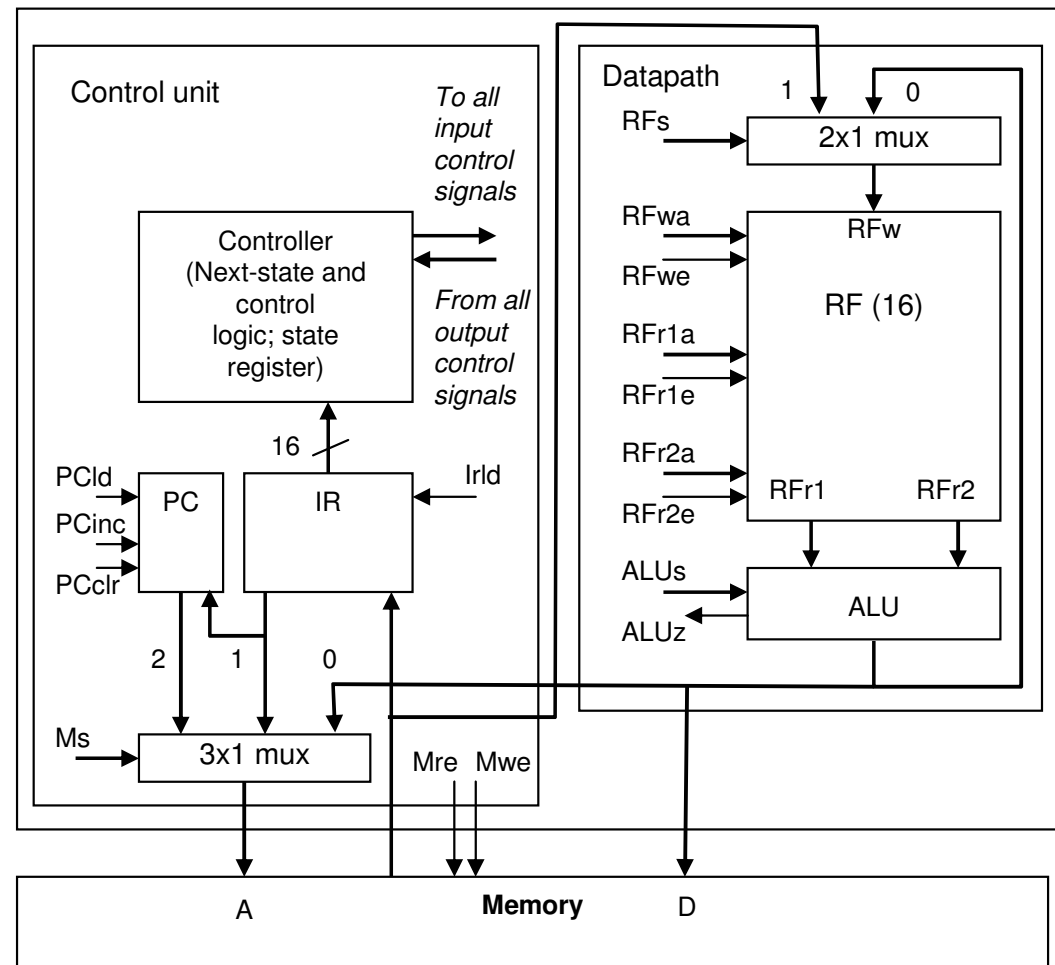
Declarations:
 bit PC[16], IR[16];
 bit M[64k][16],
 RF[16][16];

Aliases:	
op IR[15..12]	dir IR[7..0]
rn IR[11..8]	imm IR[7..0]
rm IR[7..4]	rel IR[7..0]

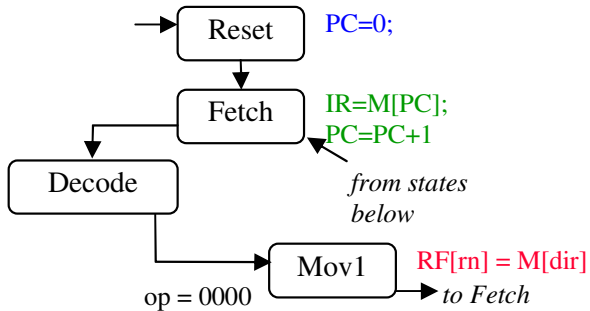


Architecture of a Simple Microprocessor

- Storage devices for each declared variable
 - register file holds each of the variables
- Functional units to carry out the FSM operations
 - One ALU carries out every required operation
- Connections added among the components' ports corresponding to the operations required by the FSM
- Unique identifiers created for every control signal



A Simple Microprocessor



$PCclr=1;$

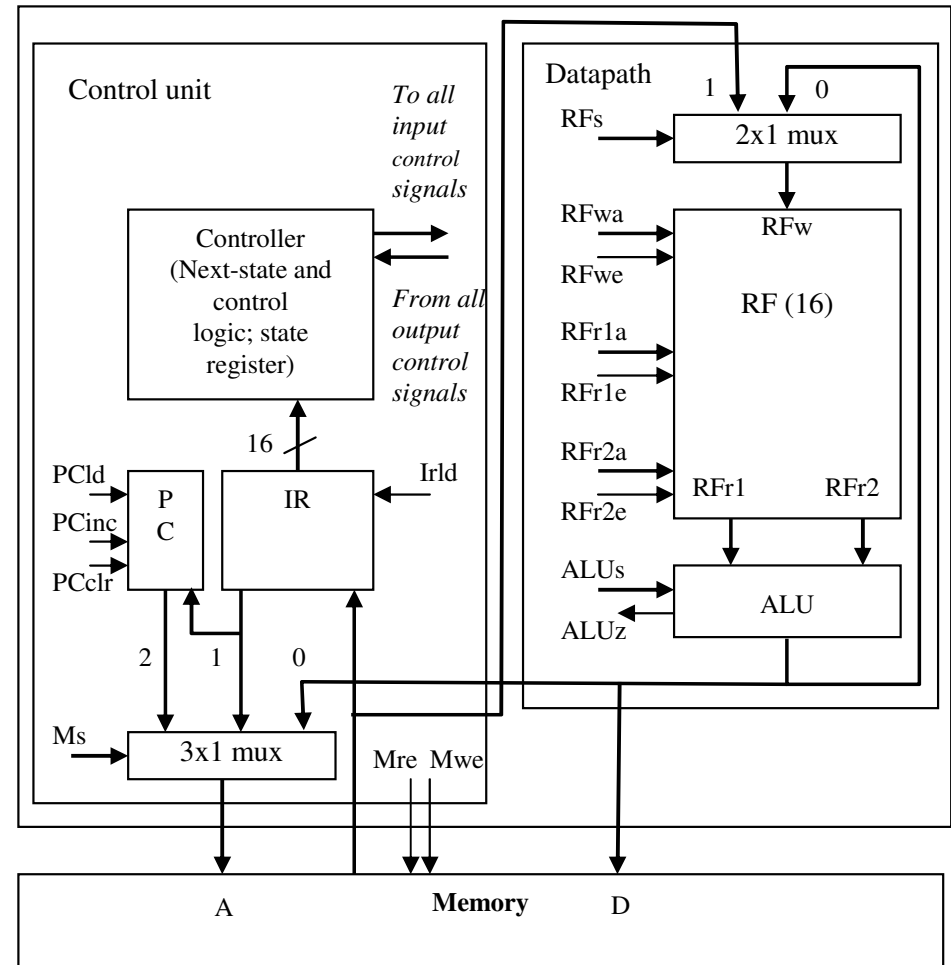
$MS=10;$
 $Ird=1;$
 $Mre=1;$
 $PCinc=1;$

$RFwa=rn;$
 $RFwe=1;$
 $RFs=01;$
 $Ms=01;$
 $Mre=1;$

Remember...

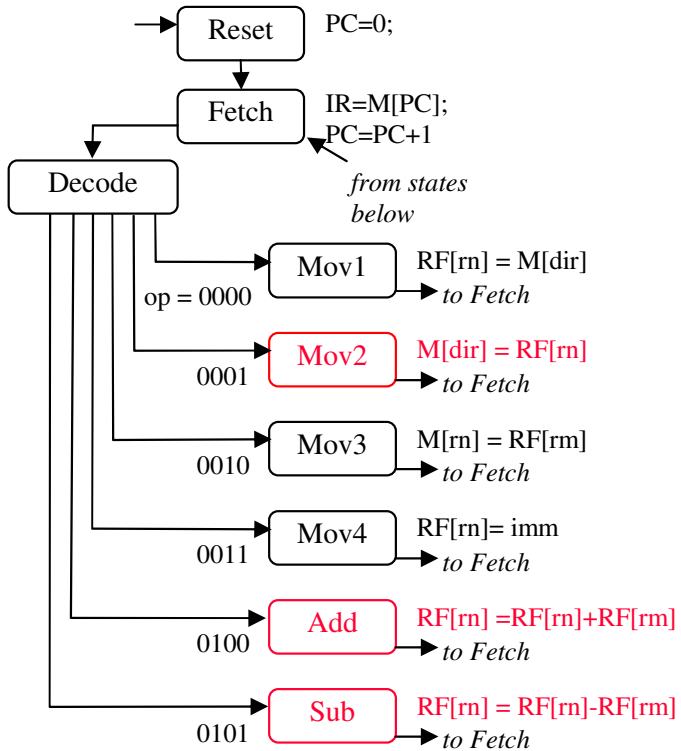
Aliases:

op	IR[15..12]	dir	IR[7..0]
rn	IR[11..8]	imm	IR[7..0]
rm	IR[7..4]	rel	IR[7..0]



You just built a simple microprocessor!

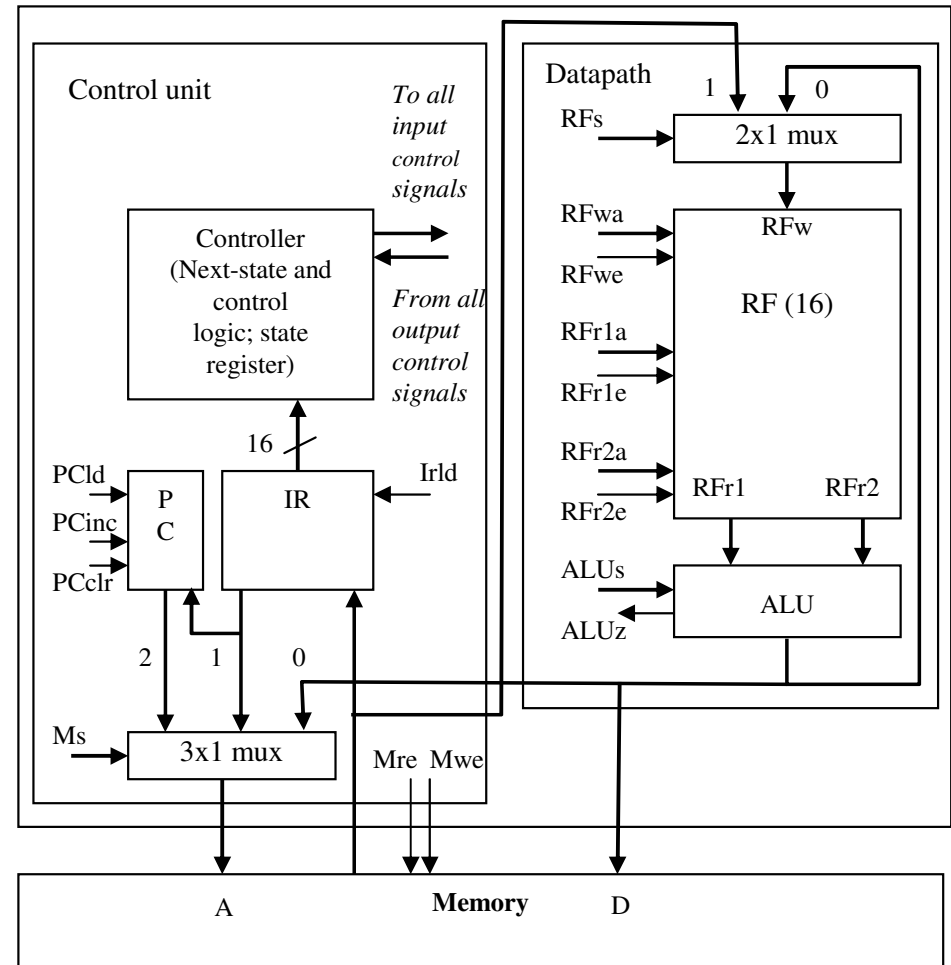
A Simple Microprocessor



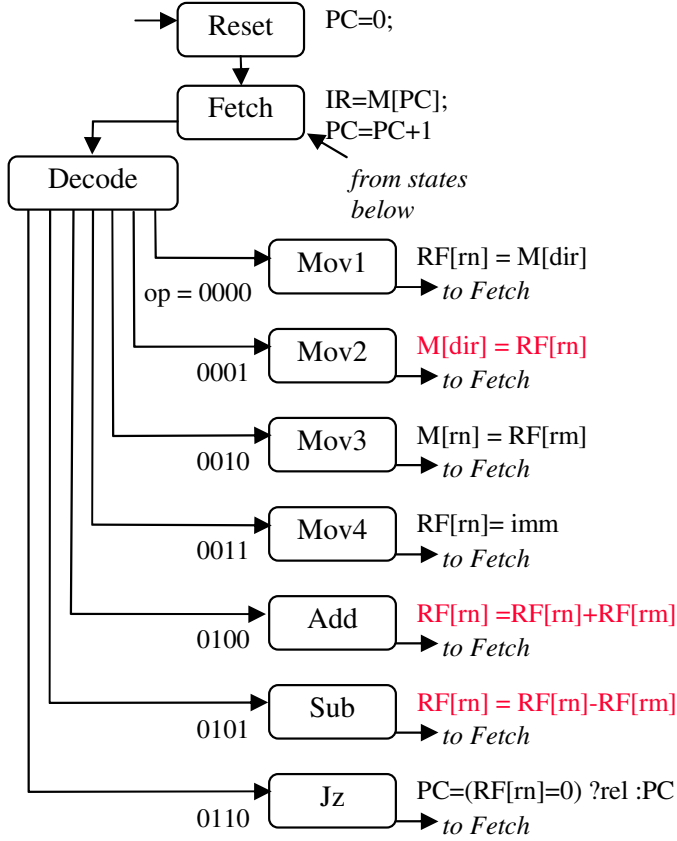
What would the rest be?

if ALUs=00 add

if ALUs=01 sub



A Simple Microprocessor



FSMD

$PCclr=1;$

$MS=10;$
 $IRld=1;$
 $Mre=1;$
 $PCinc=1;$

$RFwa=rn; RFwe=1; RFs=01;$
 $Ms=01; Mre=1;$

$RFr1a=rn; RFr1e=1;$
 $Ms=01; Mwe=1;$

$RFr1a=rn; RFr1e=1;$
 $Ms=10; Mwe=1;$

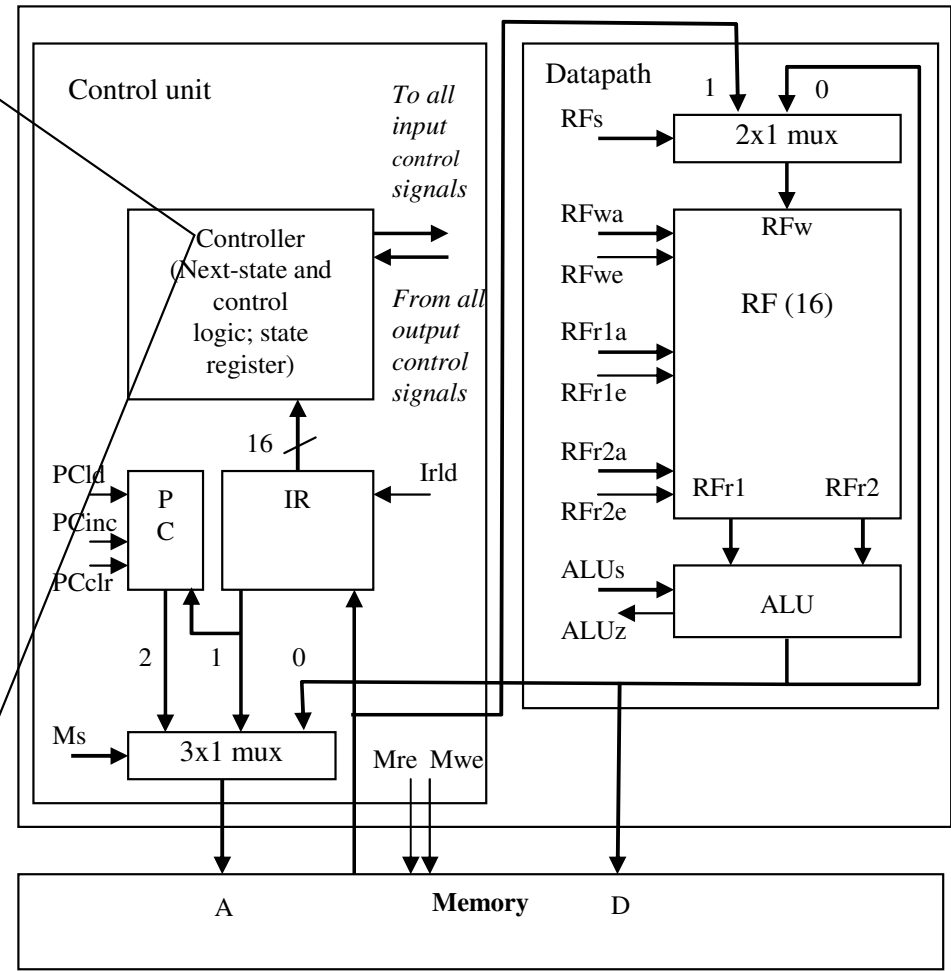
$RFwa=rn; RFwe=1; RFs=10;$

$RFwa=rn; RFwe=1; RFs=00;$
 $RFr1a=rn; RFr1e=1;$
 $RFr2a=rn; RFr2e=1; ALUs=00$

$RFwa=rn; RFwe=1; RFs=00;$
 $RFr1a=rn; RFr1e=1;$
 $RFr2a=rn; RFr2e=1; ALUs=01$

$PCld=ALUz;$
 $RFr1a=rn;$
 $RFr1e=1;$

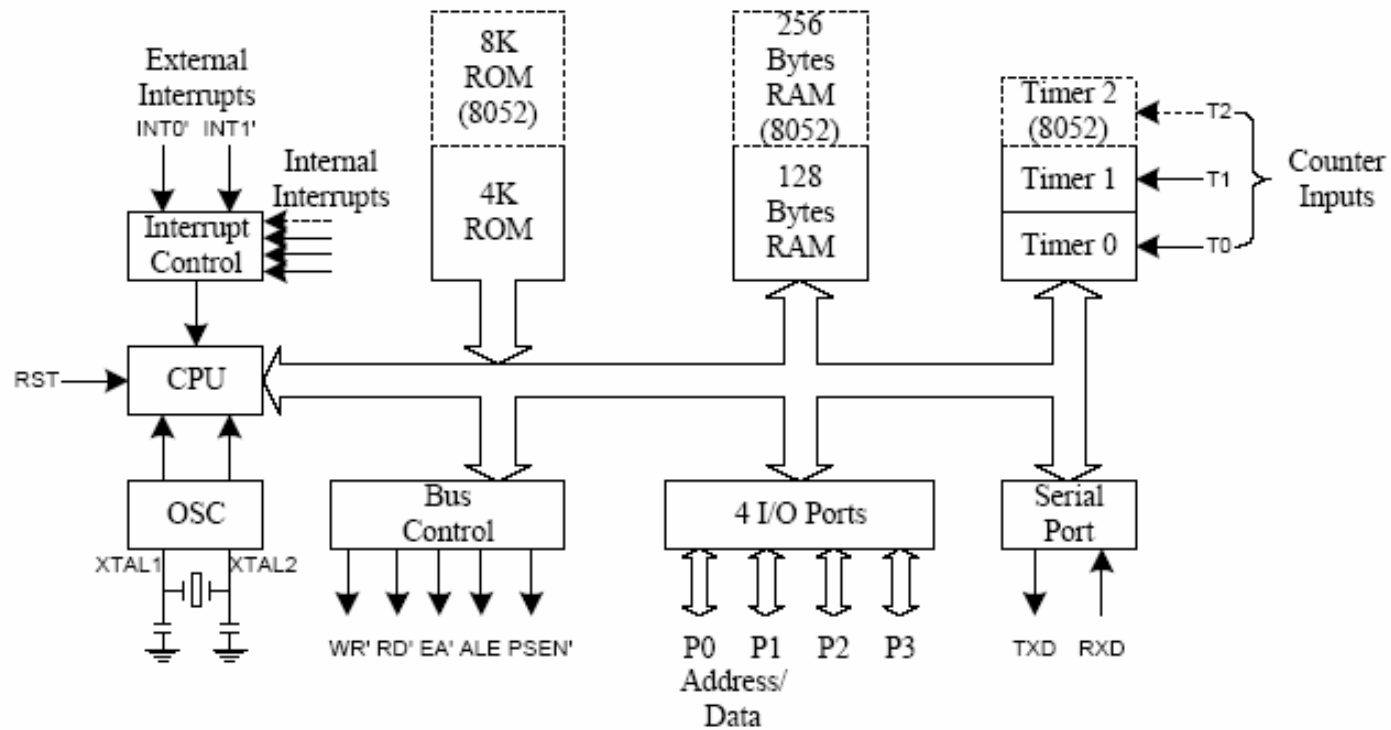
FSM operations that replace the FSMD operations after a datapath is created



You just built a simple microprocessor!

8051 specifically

8051 Block diagram



8051 Instruction Set (256 instructions!)

- **ACALL** - Absolute Call
- **ADD, ADDC** - Add Accumulator (With Carry)
- **AJMP** - Absolute Jump
- **ANL** - Bitwise AND
- **CJNE** - Compare and Jump if Not Equal
- **CLR** - Clear Register
- **CPL** - Complement Register
- **DA** - Decimal Adjust
- **DEC** - Decrement Register
- **DIV** - Divide Accumulator by B
- **DJNZ** - Decrement Register and Jump if Not Zero
- **INC** - Increment Register
- **JB** - Jump if Bit Set
- **JBC** - Jump if Bit Set and Clear Bit
- **JC** - Jump if Carry Set
- **JMP** - Jump to Address
- **JNB** - Jump if Bit Not Set
- **JNC** - Jump if Carry Not Set
- **JNZ** - Jump if Accumulator Not Zero
- **JZ** - Jump if Accumulator Zero
- **JZ** - Jump if Accumulator Zero
- **LCALL** - Long Call
- **LJMP** - Long Jump
- **MOV** - Move Memory
- **MOVC** - Move Code Memory
- **MOVX** - Move Extended Memory
- **MUL** - Multiply Accumulator by B
- **NOP** - No Operation
- **ORL** - Bitwise OR
- **POP** - Pop Value From Stack
- **PUSH** - Push Value Onto Stack
- **RET** - Return From Subroutine
- **RETI** - Return From Interrupt
- **RL** - Rotate Accumulator Left
- **RLC** - Rotate Accumulator Left Through Carry
- **RR** - Rotate Accumulator Right
- **RRC** - Rotate Accumulator Right Through Carry
- **SETB** - Set Bit
- **SJMP** - Short Jump
- **SUBB** - Subtract From Accumulator With Borrow
- **SWAP** - Swap Accumulator Nibbles
- **XCH** - Exchange Bytes
- **XCHD** - Exchange Digits
- **XRL** - Bitwise Exclusive OR

From
<http://www.win.tue.nl/~aeb/comp/8051/set8051.html>

8051 Instruction Set (256 instructions!)

- **8051 Instruction Set: DIV**
- **Operation:DIV** **Function:**Divide Accumulator by B

Instructions	OpCode	Bytes	Flags
DIV AB	0x84	1	C,OV

Description: Divides the unsigned value of the Accumulator by the unsigned value of the "B" register. The resulting quotient is placed in the Accumulator and the remainder is placed in the "B" register.

- The **Carry flag (C)** is always cleared.
- The **Overflow flag (OV)** is set if division by 0 was attempted, otherwise it is cleared.

Chapter Summary

- General-purpose processors
 - Good performance, low NRE, flexible
- Controller, datapath, and memory
- Structured languages prevail
 - But some assembly level programming still necessary
- Many tools available
 - Including instruction-set simulators, and in-circuit emulators
- ASIPs
 - Microcontrollers, DSPs, network processors, more customized ASIPs
- Choosing among processors is an important step
- Designing a general-purpose processor is conceptually the same as designing a single-purpose processor

To Do Before Next Class...

- Get to work on HW3 if you haven't already.
- Read up to page 87. The midterm will be over everything we have covered so far and what we will cover in the first half of class on Tuesday.
- The second half of class on Tuesday will be for review