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
    - Another solution: retargettable compilers
      - e.g., www.improvsys.com (customized VLIW architectures)

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*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

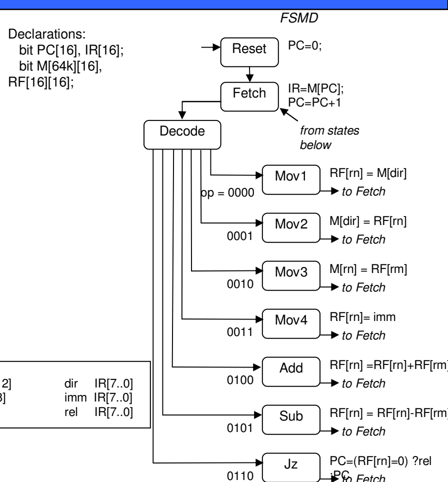
General Purpose Processors

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***

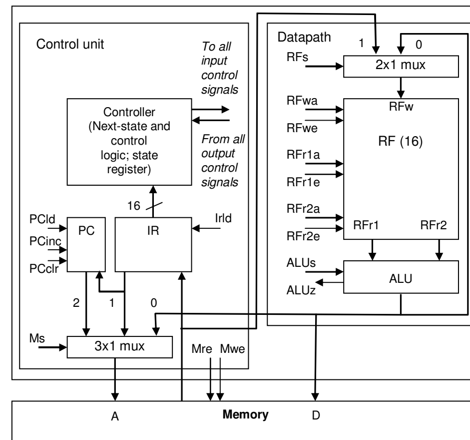
Designing a General Purpose Processor

- 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

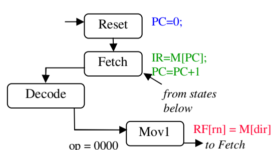


Architecture of a Simple Microprocessor

- Storage devices for each declared variable
  - register file holds each of the variables
- Functional units to carry out the FSMD 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



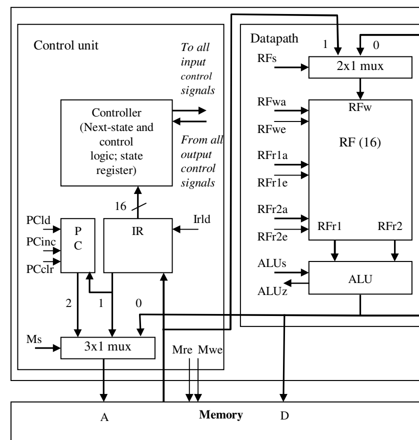
PCclr=1;

MS=10;
Irld=1;
Mre=1;
PCinc=1;

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

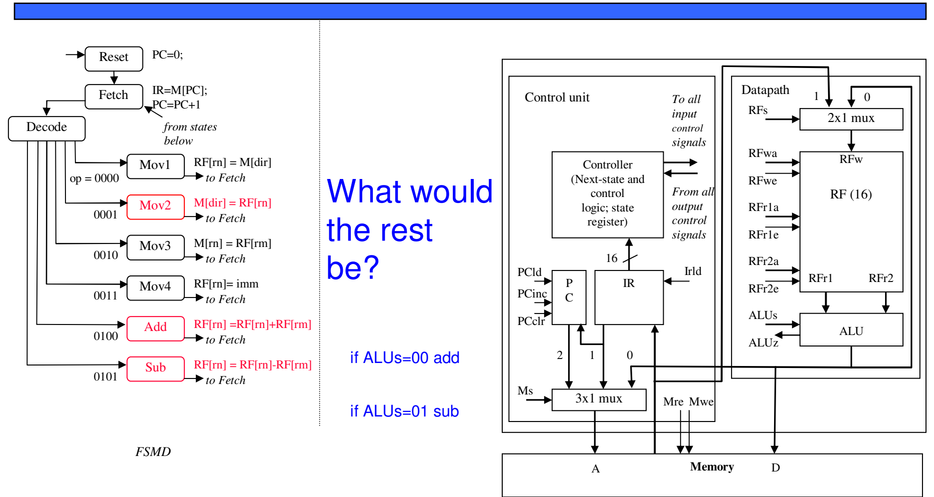
Remember…

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]

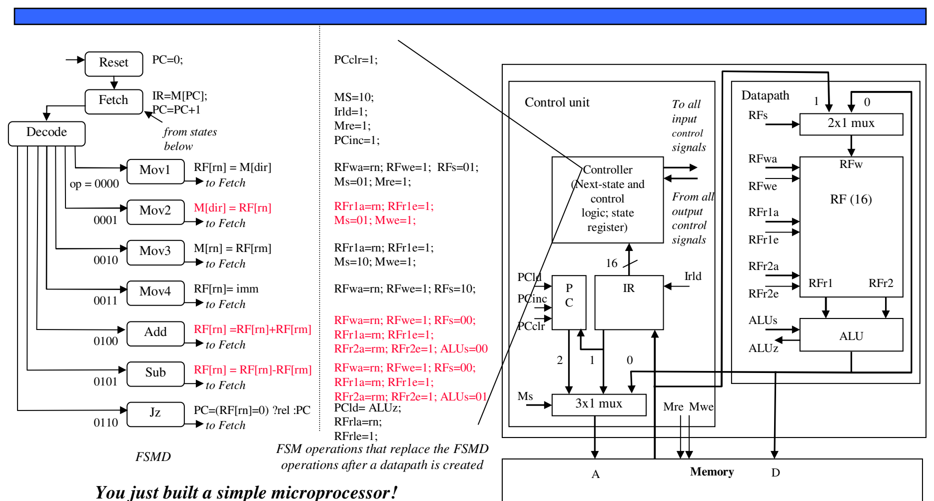


You just built a simple microprocessor!

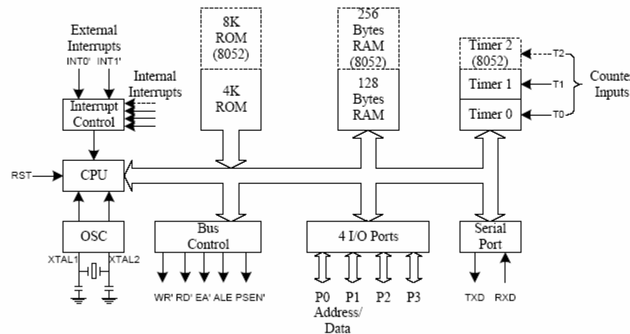
A Simple Microprocessor



A Simple Microprocessor



8051 specifically



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

Chapter 4

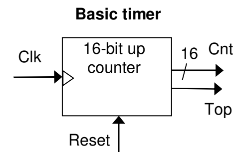
Standard Single-purpose processors

Introduction

- Single-purpose processors
  - Performs specific computation task
  - Custom single-purpose processors
    - Designed by us for a unique task
  - *Standard* single-purpose processors
    - "Off-the-shelf" -- pre-designed for a common task
    - a.k.a., peripherals
    - serial transmission
    - analog/digital conversions

Timers, counters, watchdog timers

- Timer: measures time intervals
  - To generate timed output events
    - e.g., hold traffic light green for 10 s
  - To measure input events
    - e.g., measure a car's speed
- Based on counting clock pulses
    - E.g., let Clk period be 10 ns
    - And we count 20,000 Clk pulses
    - Then 200 microseconds have passed
    - 16-bit counter would count up to 65,535*10 ns = 655.35 microsec., resolution = 10 ns
    - Top: indicates top count reached, wrap-around

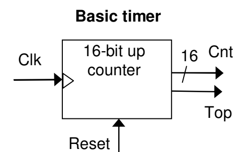


Timers, counters, watchdog timers

- Timer: measures time intervals
  - To generate timed output events
    - e.g., hold traffic light green for 10 s
  - To measure input events
    - e.g., measure a car's speed
- Based on counting clock pulses
    - E.g., let Clk period be 10 ns
    - And we count 20,000 Clk pulses
    - Then 200 microseconds have passed
    - 16-bit counter would count up to 65,535*10 ns = 655.35 microsec., resolution = 10 ns
    - Top: indicates top count reached, wrap-around

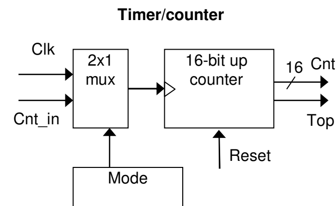
Resolution

Range



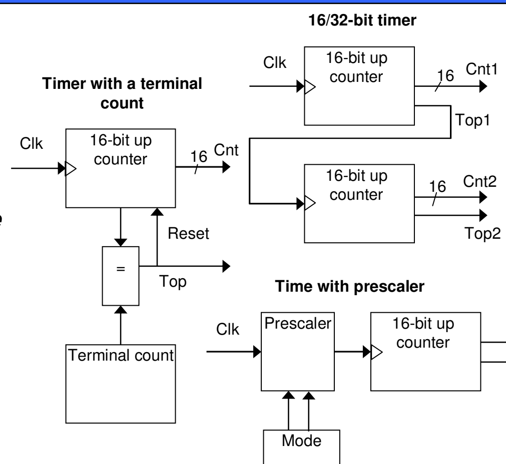
Counters

- Counter: like a timer, but counts pulses on a general input signal rather than clock
  - e.g., count cars passing over a sensor
  - Can often configure device as either a timer or counter



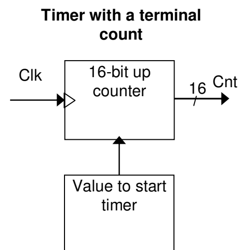
Counter/Timer structures

- Interval timer
  - Indicates when desired time interval has passed
  - We set terminal count to desired interval
    - *Number of clock cycles = Desired time interval / Clock period*
- Cascaded counters
- Prescaler
  - Divides clock
  - Increases range, decreases resolution

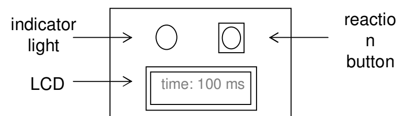


Counter/Timer structures

- Timer with ability to be initialized.



Example: Reaction Timer



- Measure time between turning light on and user pushing button
  - 16-bit timer, clk period is 83.33 ns, counter increments every 6 cycles
  - Resolution = 6*83.33=0.5 microsec.
  - Range = 65535*0.5 microseconds = 32.77 milliseconds
  - Want program to count millisec., so initialize counter to 65535 – 1000/0.5 = 63535

```
/* main.c */

#define MS_INIT      63535
void main(void){
   int count_milliseconds = 0;

   configure timer mode
   set Cnt to MS_INIT

   wait a random amount of time
   turn on indicator light
   start timer

while (user has not pushed reaction button){
   if(Top) {
      stop timer
      set Cnt to MS_INIT
      start timer
      reset Top
      count_milliseconds++;
   }
}
turn light off
printf("time:  %i ms", count_milliseconds);
}
```

Watchdog timer

A watchdog timer is similar to a counter with a terminal count, but it is used to “watch” over the system, so that if the watchdog is not reset frequently enough it will indicate that something it wrong.

Watchdog timer

Clk

16-bit up counter

Reset (kick the watchdog)

=

RESET SYSTEM

Terminal count

Serial Transmission Using UARTs

- UART: Universal Asynchronous Receiver Transmitter
  - Takes parallel data and transmits serially
  - Receives serial data and converts to parallel
- Parity: extra bit for simple error checking
- Start bit, stop bit
- Baud rate
  - signal changes per second

embedded device

1 0 0 1 1 0 1 1

1 0 0 1 1 0 1 1

Sending UART

1 0 0 1 1 0 1 1

Receiving UART

start bit

data

end bit

1 0 0 1 1 0 1 1

Parity Bit

The parity bit is for *simple* error checking.

If your 8 bit data is:

00110101

than the encoded value of 9 bit ODD parity is:

001101011

Parity bit

Your receiver will have to know that the transmitter is encoding data in 9 bit odd parity. Then it will read in 9 bit, check to make sure there are an odd number of one's and then save the data. If there are an even number of one's it will know there was an error, and either toss the data, or request a resend.

Other more complex error checking methods include CRC, and 8B/10B

Midterm Review

- Chapter 1:
  - Definitions:
    - Design Metrics (NRE, Unit Cost, Latency, Throughput, and many more)
    - Design Productivity Gap
    - Mythical Man-Month
  - Processor Technology: General, App. Specific, Single
    - Know the advantages and disadvantages of each
  - IC technology: Full Custom, Semi-Custom, PLD
  - Moore's Law

Midterm Review

- Chapter 2:
  - Custom Single Purpose Processors
  - Know Combinational Logic vs. Sequential Logic and timing
  - Know what transistors do, and how to make inverters, nands and nors
  - Remember Karnaugh Maps
  - **Know every step in creating a single purpose processor from an algorithm**
  - Know how to optimize SPP's at every level

Midterm Review

- Chapter 3:
  - General Purpose Processors
  - Know the basic parts of a processor
    - PC, IR, Controller, Datapath (ALU, Registers), main memory
  - Know how data moves in a GPP
  - Know some performance improvers (pipelining, multiple ALU's, etc.)
  - Know Harvard vs. Princeton memory arch.
  - Know what a cache is
  - Understand the different addressing modes
  - Know how to create a program in assembly
  - Know the development environment and testing terms

Midterm Review

- Chapter 4:
  - Understand counters and timers and the terms associated with them.
  - Understand Parity Bits.