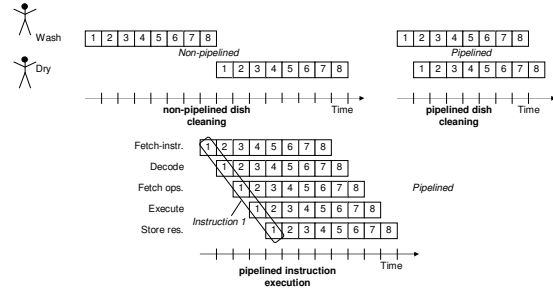


Chapter 3 General-Purpose Processors: Software

Pipelining: Increasing Instruction Throughput

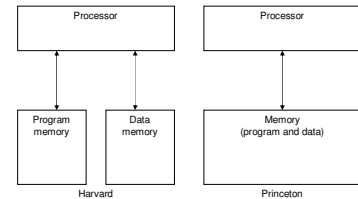


Superscalar and VLIW Architectures

- Performance can be improved by:
 - Faster clock (but there's a limit)
 - Pipelining: slice up instruction into stages, overlap stages
 - Multiple ALUs to support more than one instruction stream
 - Superscalar
 - Fetches instructions in batches, executes as many as possible
 - May require extensive hardware to detect independent instructions
 - VLIW: each word in memory has multiple independent instructions
 - Relies on the compiler to detect and schedule instructions
 - Currently growing in popularity

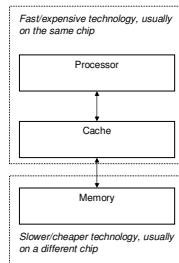
Two Memory Architectures

- Princeton
 - Fewer memory wires
- Harvard
 - Simultaneous program and data memory access



Cache Memory

- Memory access may be slow
- Cache is small but fast memory close to processor
 - Holds copy of part of memory
 - Hits and misses



Programmer's View

- Programmer doesn't need detailed understanding of architecture
 - Instead, needs to know what instructions can be executed
- Two levels of instructions:
 - Assembly level
 - Structured languages (C, C++, Java, etc.)
- Most development today done using structured languages
 - But, some assembly level programming may still be necessary
 - Drivers: portion of program that communicates with and/or controls (drives) another device
 - Often have detailed timing considerations, extensive bit manipulation
 - Assembly level may be best for these

Assembly-Level Instructions

Instruction 1	opcode	operand1	operand2
Instruction 2	opcode	operand1	operand2
Instruction 3	opcode	operand1	operand2
Instruction 4	opcode	operand1	operand2
	...		

- **Instruction Set**
 - Defines the legal set of instructions for that processor
 - Data transfer: memory/register, register/register, I/O, etc.
 - Arithmetic/logical: move register through ALU and back
 - Branches: determine next PC value when not just PC+1

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

7

A Simple (Trivial) Instruction Set

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

opcode
operands

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

8

LC-3 Instruction Set

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD ⁺	0001		DR		SR1		0	00								SR2
ADD ⁺	0001		DR		SR1		1									imm5
AND ⁺	0101		DR		SR1		0	00								SR2
AND ⁺	0101		DR		SR1		1									imm5
BR	0000		n		z		p									PCoffset9
JMP	1100				000											BaseR
JSR	0100				1											PCoffset11
JSRR	0100		0		00											BaseR
LD ⁺	0010				DR											PCoffset9
LDJ ⁺	1010				DR											PCoffset9

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

9

LC-3 Instruction Set

LDR ⁺	0110	DR	BaseR	offset6
LEA ⁺	1110	DR		PCoffset9
NOT ⁺	1001	DR	SR	111111
RET	1100	000	111	000000
RTI	1000			0000000000
ST	0011	SR		PCoffset9
STI	1011	SR		PCoffset9
STR	0111	SR	BaseR	offset6
TRAP	1111	0000		trapvect8
reserved	1101			

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

10

Addressing Modes

Addressing mode	Operand field	Register-file contents	Memory contents
Immediate	Data		
Register-direct	Register address	Data	
Register indirect	Register address	Memory address	Data
Direct	Memory address		Data
Indirect	Memory address		Memory address Data

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

11

Sample Programs

C program	Equivalent assembly program
	0 MOV R0, #0; // total = 0
	1 MOV R1, #10; // i = 10
	2 MOV R2, #1; // constant 1
	3 MOV R3, #0; // constant 0
	Loop: JZ R1, Next; // Done if i=0
	5 ADD R0, R1; // total += i
	6 SUB R1, R2; // i--
	7 JZ R3, Loop; // Jump always
	Next; // next instructions...
int total = 0; for (int i=10; i!=0; i--) total += i; // next instructions...	

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

12

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(direct)
MOV direct, Rn	0001 Rn	direct	M(direct) = Rn
MOV @Rn, Rm	0010 Rn	Rm	M(Rn) = Rm
MOV Rn, #immed.	0011 Rn	immediate	Rn = 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)

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

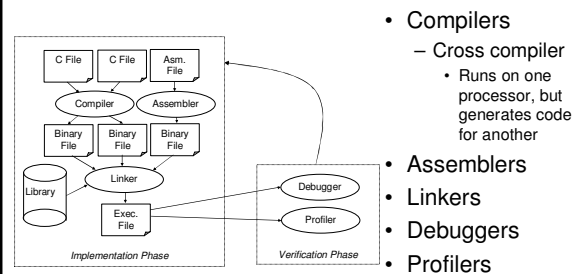
- Optional software layer providing low-level services to a program (application).
 - File management, disk access
 - Keyboard/display interfacing
 - Scheduling multiple programs for execution
- Program makes system calls to the OS

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

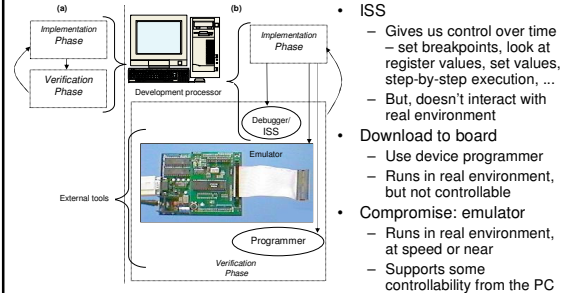


- Compilers
 - Cross compiler
 - Runs on one processor, but generates code for another
- Assemblers
- Linkers
- Debuggers
- Profilers

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 * 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 750C	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 DSP12C	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

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000
Valid/Givargis

25

To Do Before Next Class...

- Look for HW3 on the web-page.
- Read up to page 87 and start thinking about the midterm. It will be next Thursday.

Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000
Valid/Givargis

26