

CS 50011: Introduction to Systems II

Lecture 3: Computer Architecture

Prof. Jeff Turkstra



Lecture 03

- Basics
- Processors
- Architecture
- ISA
- DMA
- Modes



- Some lecture material based on:
 - Slides by Dr. George B. Adams III
 - Slides from Hennessy & Patterson
 - Slides from Silberschatz



Basics

- Moore's Law for integrated circuits
 - Transistor count for a typical processor or memory chip increases 40% to 55% per year.
 - Doubles every 18-24 months
 - Transistor count \approx computational power
- 1986-2003 computer performance increased \sim 50%/year



- 25,000-fold hardware performance improvement since 1985
 - Programs today trade execution performance for programmer productivity
 - More programming is done in managed languages like Java, Python, and C#
 - New applications have arisen: speech, sound, images, video



Moore's Law since 2003

- Microprocessor performance only 20%/year
 - Maximum power dissipation limits for air-cooled chips
 - Lack of additional instruction-level parallelism for hardware to exploit



Computer components

- Hardware
 - Transistors
 - Gates
 - Combinational and sequential circuits
 - Adders, decoders, mux/demux, latches, flip-flops, registers
 - Processors
 - Memory
 - etc



- Data types
 - Representations for character, integer, floating point, etc
 - more, od, xxd
 - Sign-magnitude
 - 1's complement
 - 2's complement
 - IEEE 754
 - BCD
 - ...



00	nul	01	soh	02	stx	03	etx	04	eof	05	enq	06	ack	07	bel
08	bs	09	ht	0A	lf	0B	vt	0C	np	0D	cr	0E	so	0F	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1A	sub	1B	esc	1C	fs	1D	gs	1E	rs	1F	us
20	sp	21	!	22	@	23	#	24	\$	25	%	26	&	27	'
28	(29)	2A	-	2B	+	2C	,	2D	-	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[5C	\	5D]	5E	^	5F	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C		7D	}	7E	~	7F	del

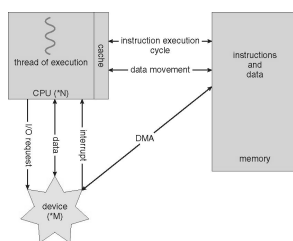
Figure 3.6 The ASCII character set. Each entry shows a hexadecimal value and the graphical representation for printable characters and the meaning for others.



- Software
 - Instructions for what to compute



How a modern computer works



Harvard architecture

- Idea by Howard Aiken, Harvard physicist, to IBM Nov. 1937
- Built by IBM in Endicott, NY and delivered to Harvard in Feb. 1944 as the Mark 1 computer
- Has separate memories for program (instructions) and data
- Input/output (I/O) to connect to the world
- Processor to carry out the computations



Harvard

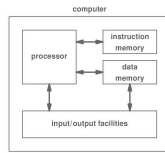


Figure 4.1 Illustration of the Harvard architecture that uses two memories, one to hold programs and another to store data.

(John) Von Neumann architecture

- Developed during his June 1945 train ride from Philadelphia to Los Alamos, NM
- He had programmed the Mark 1 in August 1944
- One memory for both data and program
- Same I/O
- Same processor

Von Neumann

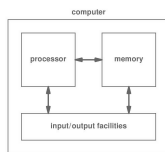


Figure 4.2 Illustration of the Von Neumann architecture. Both programs and data can be stored in the same memory.

von Neumann vs Harvard architectures

- von Neumann
 - Same memory holds instructions and data
 - Single bus between CPU and memory
 - Flexible, more cost effective
- Harvard
 - Separate memories for data and instructions
 - Two busses
 - Allows two simultaneous memory fetches
 - Less flexible, memory is physically partitioned
- Both are stored program computer designs

Processors

- Device that performs automatic computation
 - Fixed logic - single operation
 - Traffic signal sequencer
 - Selectable logic - user can select from multiple hardwired functions
 - Car with Econo and Sports modes for transmission
 - Parameterized logic - computes fixed function on variable user input
 - Programmable video recorder
 - Programmable logic processor
 - CPU, GPU, etc

Stored programs

- Some memories can be written to only once and then read many times
 - Read-Only Memory (ROM)
 - E.g., automobile engine control
- Some ROM can be re-written
 - PROM, programmable ROM
 - EPROM, erasable programmable ROM
 - EEPROM, electrically erasable...
- Embedded systems often PROM
 - Firmware upgrades

Fetch-Execute

- At the highest level, a processor does this:
repeat forever {
 FETCH, access the next program instruction from location where it is stored
 EXECUTE, perform the actions described by the instruction
}



Intel Core i7 Processor



Motherboard



Architecture basics

- Instruction set
 - Software instructions that the hardware executes
- Functional organization
 - How is the hardware partitioned into specialized units?



Architecture basics

- Logic design
 - Which logic circuits are used and how are they organized?
- Implementation
 - Technologies and packaging used



Hierarchical abstraction

- Hardware and software consist of layers in a hierarchy
 - To a good approximation
- Each layer hides (some of) its detail from the layer above
 - Principle of Abstraction
- Highest layer interacts with outside world/end user



Instruction set architecture

- Instruction set architecture (ISA) is a key level of abstraction
 - Primary interface between hardware and software
- Set of operations that a processor performs
- Instruction format defines an interpretation of bit strings
 - Similar to ASCII, 2's complement, IEEE 754, BCD, etc



Opcodes, operands, and results

- A bit string, interpreted as an instruction, specifies
 - Operations to be performed
 - Actual operand(s) and/or source(s) for the operand(s) and their type(s)
 - Destination for the result(s)



Figure 5.1 The general instruction format that many processors use. The opcode at the beginning of an instruction determines exactly which operands follow.

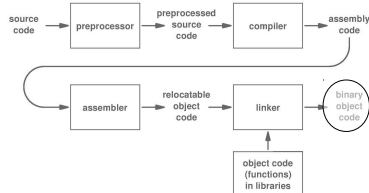


Figure 4.6 The steps used to translate a source program to the binary object code representation used by a processor.



ISA Design

- Many tradeoffs
 - Instruction length
 - Number of registers
 - Number of instructions
 - etc



CISC vs RISC

- Complex Instruction Set Computer
- Reduced Instruction Set Computer
- RISC won
 - Even Intel uses RISC micro-instructions
 - They just have a really amazing instruction decoder



Endianness

- Imagine memory is read from lowest address to highest address
- Big Endian
 - Most significant, "big," byte comes first. I.e, placed in lowest numbered memory location.
 - "Big" end appears first when reading memory
 - Network traffic
 - PowerPC, ARM, SPARC, MIPS
- Little Endian
 - Reverse of Big Endian: least significant, "Little," byte placed in lowest address
 - "Little" end first

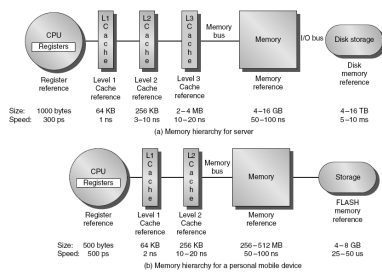


Example and comparison

- Consider $0x00C0F380 = 0x\ 00\ C0\ F3\ 80 =$
 $0b0000\ 0000\ 1100\ 0000\ 1111\ 0011\ 1000\ 0000$
 Most significant byte Least significant byte

Addresses arbitrarily start at 0x00000000; Locations accessed in arrow-indicated sequence	Byte at given location		
	Memory address	Little endian	Big endian
↓	0x00000000	1000 0000	0000 0000
	0x00000001	1111 0000	1100 0000
	0x00000002	1100 0000	1111 0011
	0x00000003	0000 0000	1000 0000

Memory hierarchy



Registers

- Type of memory located inside CPU
- Can hold a single piece of data
 - Data processing
 - Control
- Many registers
 - More later



Designing an ISA

Instruction	Meaning
add	Add the integers in two registers and place the result in a third register
load	Load an integer from the data memory into a register
store	Store the integer in a register into the data memory
jump	Jump to a new location in the instruction memory

Figure 6.1 Four example instructions, the operands each uses, and the meaning of the instruction.

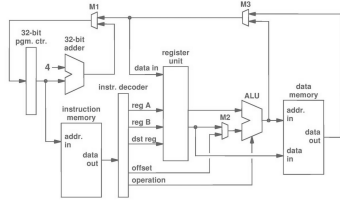


Figure 6.9 Illustration of data paths including data memory.

Example

Writing a program using these instructions is programming in assembly language; example

Assembly instr. ; Comments

load r2, 20(r1) ; r2 ← Data_Memory[20+r1]

load r3, 24(r1) ; r3 ← Data_Memory[24+r1]

add r4, r2, r3 ; r4 ← r2 + r3

store r4, 28(r1) ; Data_Memory[28+r1] ← r4

jump 60(r7) ; Fetch at Instr._Memory[60+r7]

r1, r2, r3, r4 are registers in the data path

20, 24, 28 are decimal constants

"x ←" means "x" is the location for the result

Memory[x] means the contents of memory at address x

+ means addition, with operand type defined by the instruction

(r1 + r2 is add with different data type than 28+r3)

add	operation	reg A	reg B	dst reg	unused
	0 0 0 1				
load	operation	reg A	unused	dst reg	offset
	0 0 1 0				
store	operation	reg A	reg B	unused	offset
	0 0 1 1				
jump	operation	reg A	unused	unused	offset
	0 0 1 0				

Figure 6.2 The binary representation for each of the four instructions listed in Figure 6.1. Each instruction is thirty-two bits long.

Assembly and its machine code

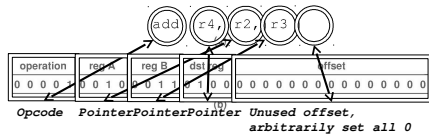
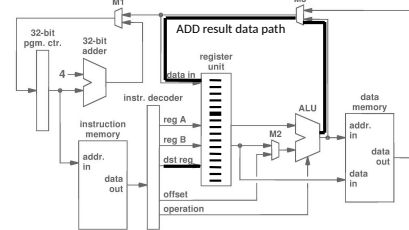


Figure 6.3 (a) An example *add* instruction as it appears to a programmer, and (b) the instruction stored in memory.

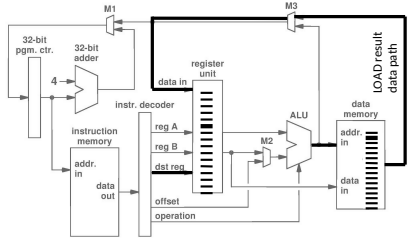
ADD



ADD: ALU output is result, deliver to dst reg; result has the meaning "integer" because this is an integer adder

Figure 6.9 Illustration of data paths including data memory.

LOAD

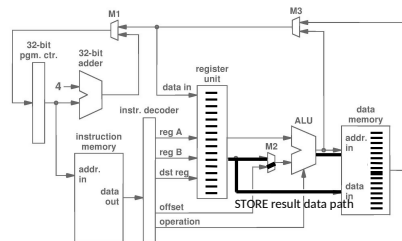


LOAD: ALU output is pointer that must be sent to data memory, which then produces copy of the location contents which, finally, must be written into dst_reg; Result is a bit string from memory; no inherent meaning at all.

© 2017 Dr. Jeffrey A. Turkstra

43

STORE

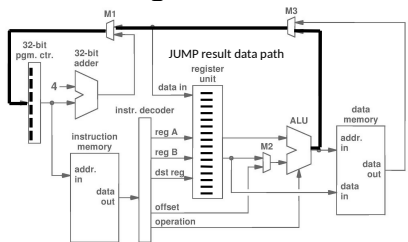


STORE: ALU output is pointer that must be sent to data memory along with the value from reg B to be written into the data memory location; Result is a bit string from reg_B written in memory; no inherent meaning at all.

© 2017 Dr. Jeffrey A. Turkstra

44

JUMP



JUMP: ALU output is computed Next_instruction_pointer, must deliver to Instruction_pointer_register; Result meaning is location of next instruction on the execution path.

© 2017 Dr. Jeffrey A. Turkstra

45

Intel Core microarchitecture pipeline

Direct memory access

- DMA allows other hardware subsystems to access main memory without going through the CPU
- Modern systems usually have DMA controller (MMU)
 - Memory address register, byte count, control, etc
 - Responsible for ensuring accesses are properly restrained
 - Attack vector

© 2017 Dr. Jeffrey A. Turkstra

47

MMU

- Responsible for "refreshing" DRAM
- Translates virtual memory addresses to physical addresses
- Sometimes part of CPU
- Sometimes not
 - Northbridge for Intel until recently
 - I7/i5 have an Integrated Memory Controller (IMC)

© 2017 Dr. Jeffrey A. Turkstra

48

Page 70



Execution modes

- CPU hardware has several possible modes
 - At any one time, in one mode
- Modes specify
 - Privilege level
 - Valid instructions
 - Valid memory addresses
 - Size of data items
 - Backwards compatibility



Rings



Ring -1

- Intel Active Management Technology
- Exists for other architectures as well
- Runs on the Intel Management Engine (ME)
 - Isolated and protected coprocessor
 - Embedded in all current Intel chipsets
 - ARC core
 - Out-of-band access
 - Direct access to Ethernet controller
- Requires vPro-enabled CPU/Motherboard/Chipset



Ring -1

- ...if you can exploit it, you win.
 - CVE-2017-5689
- Go read about it



Trusting trust

- Reflections on Trusting Trust
 - by Ken Thompson
- Read this too



How to change between modes

- Automatic
 - Hardware interrupts
 - OS-specified handlers
- “Manual”
 - Initiated by software, typically OS
 - System calls, signals, and page faults
 - Sometimes mode can be set by application



Paging and Virtual Memory

- ...later



Questions?

