

Lecture 2: The Instruction Set Architecture

COS / ELE 375

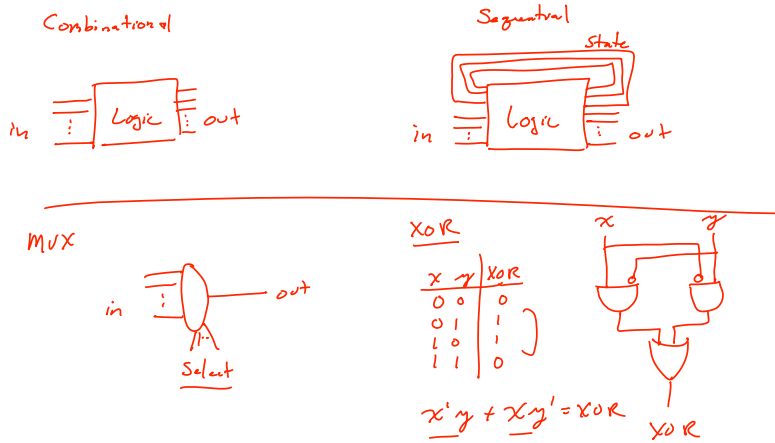
Computer Architecture and Organization

Princeton University
Fall 2015

Prof. David August

1

Quiz 0



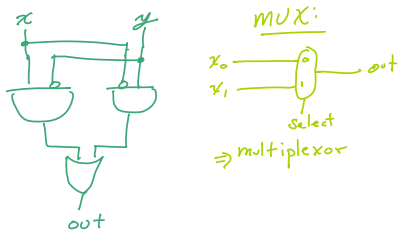
2

Quiz 0

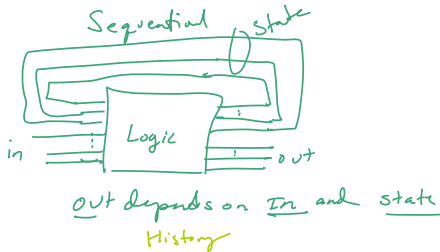
Bit: Binary Digit / two distinct states of matter or energy

XOR:

x	y	out
0	0	0
0	1	1
1	0	1
1	1	0

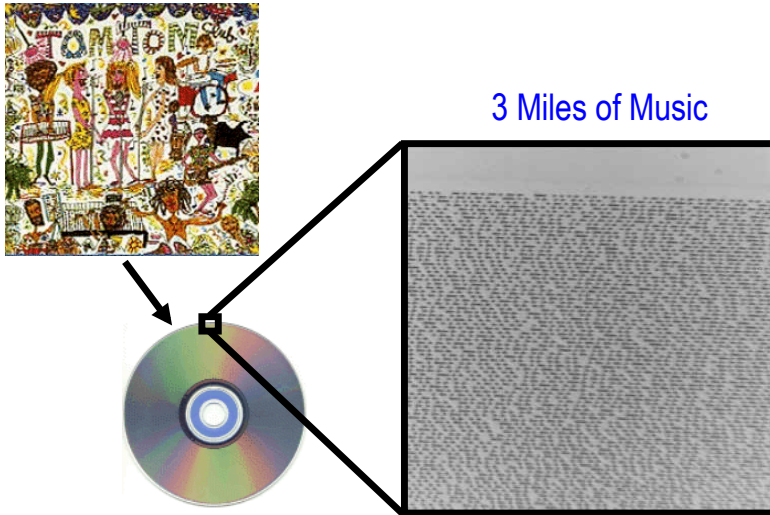


Combinational Circuit



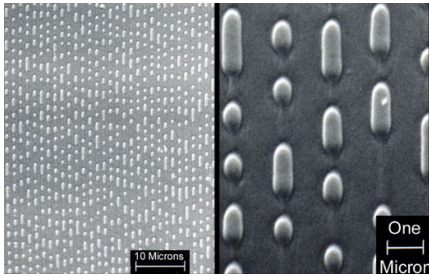
3

CD

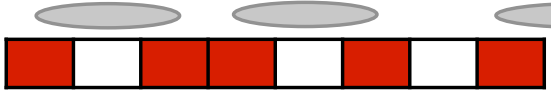


4

Pits and Lands



Transition represents a bit state (1/on/red/female/heads)
No change represents other state (0/off/white/male/tails)



5

Interpretation



As Music:

$01110101_2 = 117/256$ position of speaker

As Number:

$01110101_2 = 1 + 4 + 16 + 32 + 64 = 117_{10} = 75_{16}$

(Get comfortable with base 2, 8, 10, and 16.)

As Text:

$01110101_2 = 117^{\text{th}}$ character in the ASCII codes = "u"

6

Interpretation – ASCII

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☉	SOH	033	!	065	A	097	a
002	●	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	\$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	▲	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	■	BS	040	(072	H	104	h
009	(tab)	HT	041)	073	I	105	i
010	(line feed)	LF	042	.	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	↵	SO	046	.	078	N	110	n
015	⊙	SI	047	/	079	O	111	o
016	▼	DLE	048	0	080	P	112	p
017	↑	DC1	049	1	081	Q	113	q
018	↓	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↑	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[123	{
028	(cursor right)	FS	060	<	092	\	124	
029	(cursor left)	GS	061	=	093]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	␣

7

Princeton Computer Science Building West Wall



8

Interpretation



As Music:

$$01110101_2 = 117/256 \text{ position of speaker}$$

As Number:

$$01110101_2 = 1 + 4 + 16 + 32 + 64 = 117_{10} = 75_{16}$$

As Text:

$$01110101_2 = 117^{\text{th}} \text{ character in the ASCII codes} = \text{"u"}$$

CAN ALSO BE INTERPRETED AS MACHINE INSTRUCTION!

9

Binary Code and Data (Hello World!)

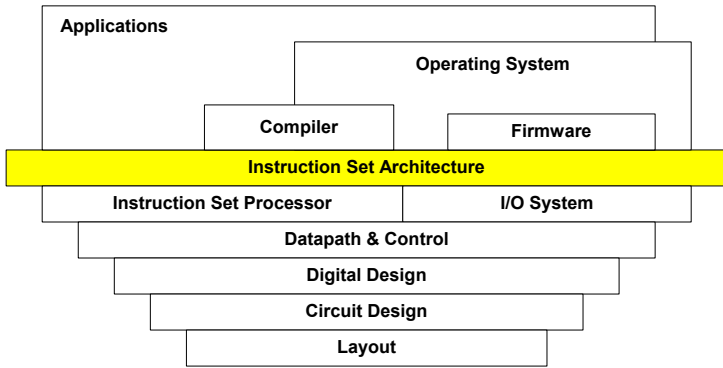
- Programs consist of Code and Data
- Code and Data are Encoded in Bits

```

00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF.....
...
00000260: 5002 0000 0000 0000 006c 6962 632e 736f  P.....libc.so
00000270: 2e36 2e31 0070 7269 6e74 6600 5f5f 6c69  .G.L.printf__ii
00000280: 6263 5f73 7461 7274 5f6d 6169 6e00 474c  bc_start_main.GL
00000290: 4942 435f 322e 3200 0000 0200 0200 0000  IB_C_2.2.....
...
00000860: 4865 6c6c 6f20 576f 726c 6421 0d00 0000  Hello world!...
...
40000000000000690 <main>:
40000000000000690: 00 10 15 08 80 05  [MII]  alloc r34=ar.pfs,5,4,0
40000000000000696: 30 02 30 00 42 20  mov r35=r12
4000000000000069c: 04 00 c4 00  mov r33=b0
400000000000006a0: 0a 20 81 03 00 24  [MII]  addl r36=96,r1;;
400000000000006a6: 40 02 90 30 20 00  ldr r36=[r36]
400000000000006ac: 04 08 00 84  mov r32=r1
400000000000006b0: 1d 00 00 00 01 00  [MFB]  nop.m 0x0
400000000000006b6: 00 00 00 02 00 00  nop.f 0x0
400000000000006bc: b8 fd ff 58  br.call.sptk.many b0=4000000000000460;;
400000000000006c0: 00 08 00 40 00 21  [MII]  mov r1=r32
400000000000006c6: 80 00 00 00 42 00  mov r8=r0
400000000000006cc: 20 02 aa 00  mov.i ar.pfs=r34
400000000000006d0: 00 00 00 00 01 00  [MII]  nop.m 0x0
400000000000006d6: 00 08 05 80 03 80  mov b0=r33
400000000000006dc: 01 18 01 84  mov r12=r35
400000000000006e0: 1d 00 00 00 01 00  [MFB]  nop.m 0x0
400000000000006e6: 00 00 00 02 00 80  nop.f 0x0
400000000000006ec: 08 00 84 00  br.ret.sptk.many b0;;
    
```

Interfaces in Computer Systems

Software: Produce Bits Instructing Machine to Manipulate State or Produce I/O

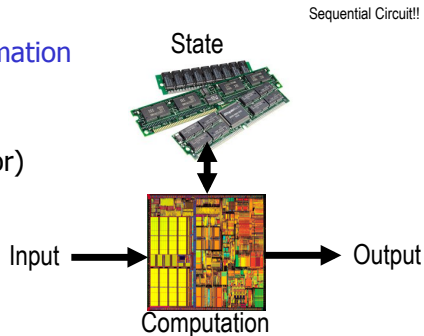


Hardware: Read and Obey Instruction Bits

Instructions

Computers process information

- Input/Output (I/O)
- State (memory)
- Computation (processor)



- Instructions instruct processor to manipulate state
- Instructions instruct processor to produce I/O in the same way

State

Typical modern machine has this **architectural** state:

1. Main Memory
2. Registers
3. Program Counter

Architectural – Part of the assembly programmer’s interface
(Implementation has additional microarchitectural state)

13

State – Main Memory

Main Memory (AKA: RAM – Random Access Memory)

- Data can be accessed by address (like a big array)
- Large but relatively slow
- Decent desktop machine: 1 Gigabyte, 800MHz

Address	Data
0000	01011001 ₂
0001	F5 ₁₆
0002	78 ₁₆
0003	3A ₁₆
...	...
FFFF	00000000 ₂



Byte Addressable

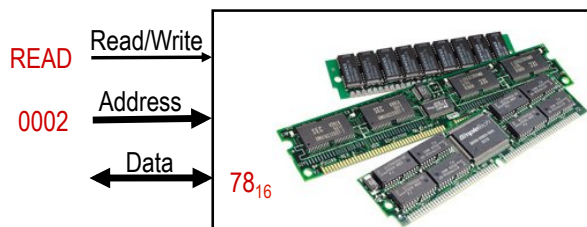
14

State – Main Memory

Read:

1. Indicate READ
2. Give Address
3. Get Data

Address	Data
0000	01011001 ₂
0001	F5 ₁₆
0002	78 ₁₆
0003	3A ₁₆
...	...
FFFF	00000000 ₂



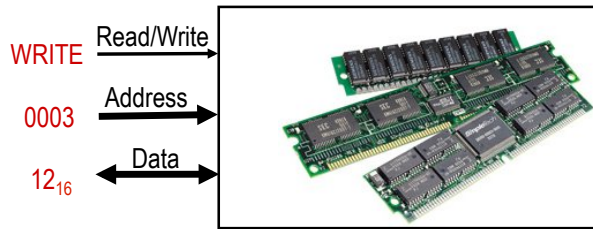
15

State – Main Memory

Write:

1. Indicate WRITE
2. Give Address and Data

Address	Data
0000	01011001 ₂
0001	F5 ₁₆
0002	78 ₁₆
0003	12 ₁₆
...	...
FFFF	00000000 ₂



16

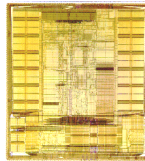
State – Registers

Registers (AKA: Register File)

- Data can be accessed by register number (address)
- Small but relatively fast (typically on processor chip)
- Decent desktop machine: 8 32-bit registers, 3 GHz



Register	Data in Reg
0	00000000 ₁₆
1	F629D9B5 ₁₆
2	7B2D9D08 ₁₆
3	00000001 ₁₆
...	...
8	DEADBEEF ₁₆

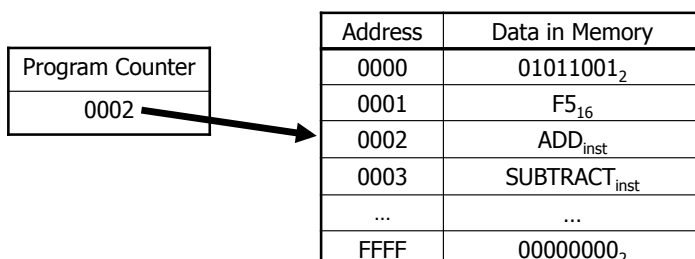


17

State – Program Counter

Program Counter (AKA: PC, Instruction Pointer, IP)

- Instructions change state, but which instruction now?
- PC holds memory address of currently executing instruction

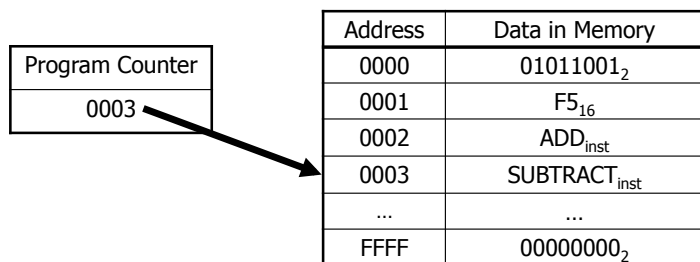


18

State – Program Counter

Program Counter (AKA: PC, Instruction Pointer, IP)

- Instructions change state, but which instruction now?
- PC holds address of currently executing instruction
- PC is updated after each instruction



19

State – Summary

Typical modern machine has this **architectural** state:

1. Main Memory – Big, Slow
2. Registers – Small, Fast (always on processor chip)
3. Program Counter – Address of executing instruction

Architectural – Part of the assembly programmer’s interface

(implementation has additional microarchitectural state)

20

An Aside: State and The Core Dump

- Core Dump: the state of the machine at a given time
- Typically at program failure
- Core dump contains:
 - Register Contents
 - Memory Contents
 - PC Value

Registers							
0	1	2	3	4	5	6	7
0000	0788	B700	0010	0401	0002	0003	00A0
8	9	A	B	C	D	E	F
0000	0788	B700	0010	0401	0002	0003	00A0

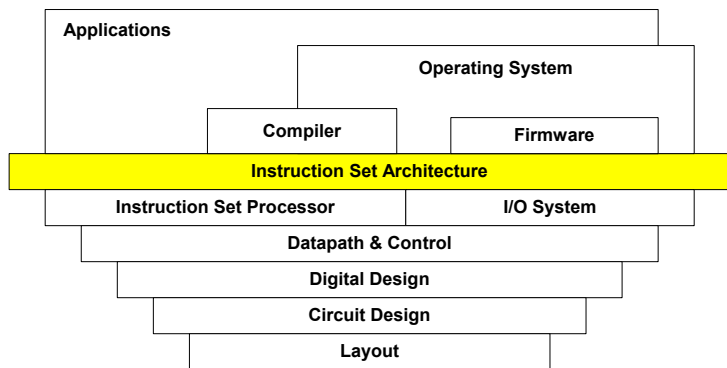
PC	10
----	----

Main Memory							
00:	0000	0000	0000	0000	0000	0000	0000
08:	0000	0000	0000	0000	0000	0000	0000
10:	9222	9120	1121	A120	1121	A121	7211 0000
18:	0000	0001	0002	0003	0004	0005	0006 0007
20:	0008	0009	000A	000B	000C	000D	000E 000F
28:	0000	0000	0000	FE10	FACE	CAFE	ACED CEDE
.							
.							
E8:	1234	5678	9ABC	DEF0	0000	0000	F00D 0000
F0:	0000	0000	EEEE	1111	EEEE	1111	0000 0000
F8:	B1B2	F1F5	0000	0000	0000	0000	0000 0000

21

Interfaces in Computer Systems

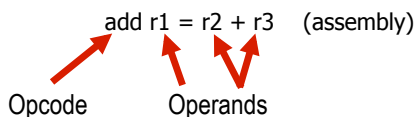
Software: Produce Bits Instructing Machine to Manipulate State or Produce I/O



Hardware: Read and Obey Instruction Bits

Instructions

An ADD Instruction:



Parts of the Instruction:

- Opcode (verb) – what operation to perform
- Operands (noun) – what to operate upon
- Source Operands – where values come from
- Destination Operand – where to deposit data values

Instructions

Instructions:

“The vocabulary of commands”
Specify how to operate on state

Example:

```
40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[ r3 ] = r1
52: load r2 = M[ 2 ]
```

Program Counter
40

Register	Data
0	0
1	15
2	1
3	2
...	...
31	0

Address	Data
0	0
1	25
2	5
3	9
...	...
FFFFFFFF	0

Instructions

Instructions:
“The vocabulary of commands”
Specify how to operate on state

Example:

40: add 3 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]

Program Counter
40

Register	Data
0	0
1	15
2	1
3	2
...	...
31	0

Address	Data
0	0
1	25
2	5
3	9
...	...
FFFFFFFF	0

Instructions

Example:

40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]

Program Counter
40

Register	Data
0	0
1	3
2	1
3	2
...	...
31	0

Address	Data
0	0
1	25
2	5
3	9
...	...
FFFFFFFF	0

Instructions

Instructions:
“The vocabulary of commands”
Specify how to operate on state

Example:

40: add r1 = r2 + r3
44: sub 3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]

Program Counter
44

Register	Data
0	0
1	3
2	1
3	2
...	...
31	0

Address	Data
0	0
1	25
2	5
3	9
...	...
FFFFFFFF	0

Instructions

Instructions:

“The vocabulary of commands”
Specify how to operate on state

Example:

40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = 3
52: load r2 = M[2]

Program Counter
48

Register	Data
0	0
1	3
2	1
3	3
...	...
31	0

Address	Data
0	0
1	25
2	5
3	9
...	...
FFFFFFFF	0

Instructions

Instructions:

“The vocabulary of commands”
Specify how to operate on state

Example:

40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load 5 = M[2]

Program Counter
52

Register	Data
0	0
1	3
2	1
3	3
...	...
31	0

Address	Data
0	0
1	25
2	5
3	3
...	...
FFFFFFFF	0

Instructions

Instructions:

“The vocabulary of commands”
Specify how to operate on state

Example:

40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]

Program Counter
52

Register	Data
0	0
1	3
2	5
3	3
...	...
31	0

Address	Data
0	0
1	25
2	5
3	3
...	...
FFFFFFFF	0

Instructions

Note:

1. Insts Executed in Order
2. Addressing Modes

Example:

```
40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[ r3 ] = r1
52: load r2 = M[ 2 ]
```

Program Counter
52

Register	Data
0	0
1	3
2	5
3	3
...	...
31	0

Address	Data
0	0
1	25
2	5
3	3
...	...
FFFFFFFF	0

Assembly Instructions and C

```
add r1 = r2 + r3
```

```
sub r3 = r1 - r0
```

```
store M[ r3 ] = r1
```

```
load r2 = M[ 2 ]
```

```
main() {
    int a = 15, b = 1, c = 2;

    a = b + c; /* a gets 3 */

    c = a; /* c gets 3 */

    *(int *)c = a;
        /* M[c] = a */

    b = *(int *) (2);
        /* b gets M[2] */
}
```



Branching

Suppose we could only execute instructions in sequence.

Recall from our example:

40: add r1 = r2 + r3; $PC = PC + 4$
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]

- In a decent desktop machine, how long would the longest program stored in main memory take?
- Assume:
 - 1 instruction per cycle
 - An instruction is encoded in 4 bytes (32 bits)

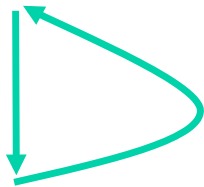
34

Therefore...

- Some instructions must execute more than once
- PC must be updated

Example:

40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]
56: PC = 40



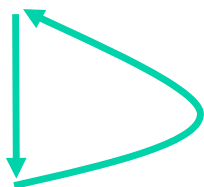
35

Unconditional Branches

- Unconditional branches always update the PC
- AKA: Jump instructions

Example:

40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[r3] = r1
52: load r2 = M[2]
56: jump 40



- How long with the program take?

36

Conditional Branch

- Conditional Branch sometimes updates PC
- AKA: Branch, Conditional Jump
- Example

```
40: r1 = 10
44: r1 = r1 - 1
48: branch r1 > 0, 44 ← if r1 is greater than 0, PC = 44
52: halt
```
- How long will this program take?

37

Conditional Branch

- What does this look like in C?
- Example

```
10: "Hello\n"      ; data in memory
36: arg1 = 10     ; argument memory address is 10
40: r1 = 10
44: r1 = r1 - 1
48: call printf   ; printf(arg1)
52: branch r1 > 0, 44
56: halt
```

Details about red instructions/data next time...

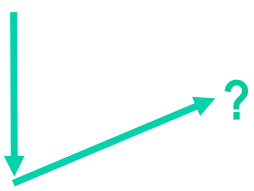
38

Indirect Branches

- Branch address may also come from a register
- AKA: Indirect Jump

Example:

```
40: add r1 = r2 + r3
44: sub r3 = r1 - r0
48: store M[ r3 ] = r1
52: load r2 = M[ 2 ]
56: jump r4
60: halt
```



39

Branch Summary

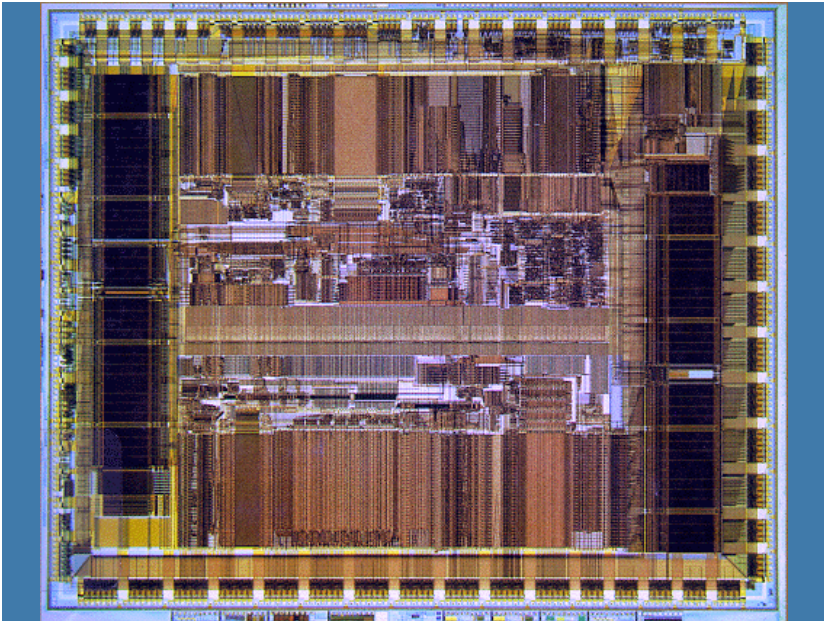
- Reduce, Reuse, Recycle (instructions)
- Branch instructions update state

Registers							
0	1	2	3	4	5	6	7
0000	0788	B700	0010	0401	0002	0003	00A0
8	9	A	B	C	D	E	F
0000	0788	B700	0010	0401	0002	0003	00A0

PC
10

Main Memory							
00:	0000	0000	0000	0000	0000	0000	0000
08:	0000	0000	0000	0000	0000	0000	0000
10:	9222	9120	1121	A120	1121	A121	7211 0000
18:	0000	0001	0002	0003	0004	0005	0006 0007
20:	0008	0009	000A	000B	000C	000D	000E 000F
28:	0000	0000	0000	FE10	FACE	CAFE	ACED CEDE
.							
.							
EB:	1234	5678	9ABC	DEFO	0000	0000	F00D 0000
F0:	0000	0000	EEEE	1111	EEEE	1111	0000 0000
FB:	B1B2	F1F5	0000	0000	0000	0000	0000 0000

40



A Note on Notation...

- Assembly syntax is somewhat arbitrary
- Equivalent “Add” Instructions
 - add r1, r2, r3
 - add r1 = r2, r3
 - $r1 = r2 + r3$
 - add r1 = r2 + r3
 - add \$1, \$2, \$3
 - ...
- Equivalent “Store Word” Instructions
 - sw \$1, 10(\$2)
 - $M[r2 + 10] = r1$
 - st.w $M[r2 + 10] = r1$
 - ...

42

Specific Instance: MIPS Instruction Set

- MIPS – SGI Workstations, Nintendo, Sony...

State:

- 32-bit addresses to memory (32-bit PC)
- 32 32-bit Registers
- A “word” is 32-bits on MIPS
- Register \$0 (\$zero) always has the value 0
- By convention, certain registers are used for certain things – more next time...

43

MIPS Usage in Course

- Used throughout book
- We will use it on homework and exams
- For clarity of lecture, MIPS not always used
- Refer to book for all instructions discussed

44

Specific Instance: MIPS Instruction Set

Some Arithmetic Instructions:

- Add:
 - Assembly Format: add <dest>, <src1>, <src2>
 - Example: add \$1, \$2, \$3
 - Example Meaning: $r1 = r2 + r3$
- Subtract:
 - Same as add, except “sub” instead of “add”

45

Specific Instance: MIPS Instruction Set

Some Memory Instructions:

- Load Word:
 - Assembly Format: lw <dest>, <offset immediate> (<src1>)
 - Example: lw \$1, 100 (\$2)
 - Example Meaning: $r1 = M[r2 + 100]$
- Store Word:
 - Assembly Format: sw <src1>, <offset immediate> (<src2>)
 - Example: sw \$1, 100 (\$2)
 - Example Meaning: $M[r2 + 100] = r1$

46

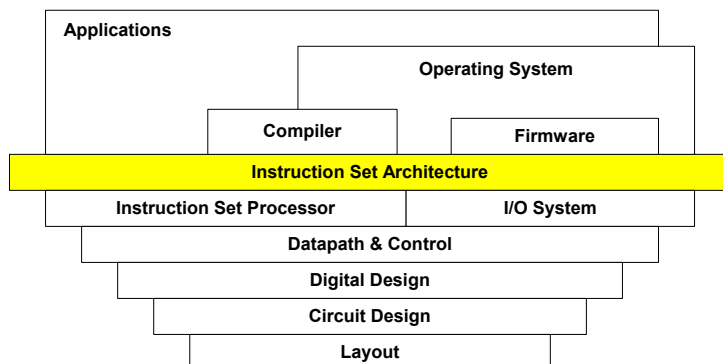
Specific Instance: MIPS Instruction Set

Some Branch Instructions:

- Branch Equal:
 - Assembly Format: beq <src1>, <src2>, <target immediate>
 - Example: beq \$1, \$2, 100
 - Example Meaning: branch $r1 == r2, 100$
If $r1$ is equal to $r2$, $PC = 100$
- Branch Not Equal: Same except beq -> bne
- Jump:
 - Assembly Format: j <target immediate>
 - Example: j 100
 - Example Meaning: jump 100
 $PC = 100$

47

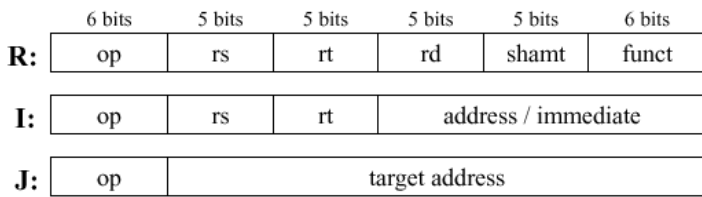
How are MIPS Instructions Encoded?



48

MIPS Encodings

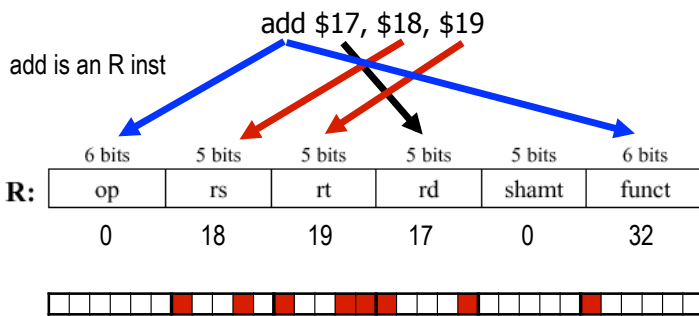
32-bits/Instruction



op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode (function code)
address: offset for load/store instructions ($\pm 2^{15}$)
immediate: constants for immediate instructions

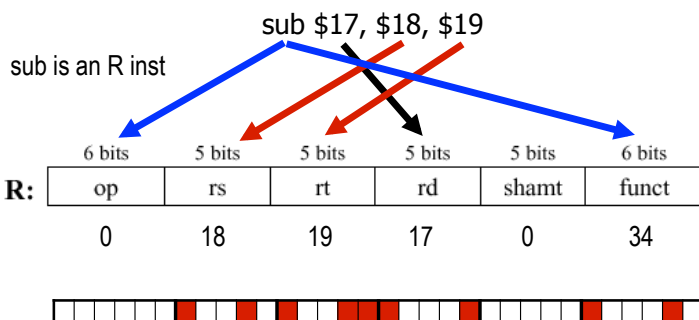
49

MIPS Add Instruction Encoding



50

MIPS Add Instruction Encoding



51

Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data

Registers hold 32 bits of data

- 2^{32} bytes with byte addresses from 0 to $2^{32}-1$
- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$
- Words are aligned
i.e., what are the least 2 significant bits of a word address?

Addressing Modes

Addressing mode	Example	Meaning
Register	Add R4,R3	$R4 \leftarrow R4+R3$
Immediate	Add R4,#3	$R4 \leftarrow R4+3$
Displacement	Add R4,100(R1)	$R4 \leftarrow R4+\text{Mem}[100+R1]$
Register indirect	Add R4,(R1)	$R4 \leftarrow R4+\text{Mem}[R1]$
Indexed / Base	Add R3,(R1+R2)	$R3 \leftarrow R3+\text{Mem}[R1+R2]$
Direct or absolute	Add R1,(1001)	$R1 \leftarrow R1+\text{Mem}[1001]$
Memory indirect	Add R1,@(R3)	$R1 \leftarrow R1+\text{Mem}[\text{Mem}[R3]]$
Auto-increment	Add R1,(R2)+	$R1 \leftarrow R1+\text{Mem}[R2]; R2 \leftarrow R2+d$
Auto-decrement	Add R1,-(R2)	$R2 \leftarrow R2-d; R1 \leftarrow R1+\text{Mem}[R2]$
Scaled	Add R1,100(R2)[R3]	$R1 \leftarrow R1+\text{Mem}[100+R2+R3*d]$

56

Hello World

The Hello World Algorithm:

1. Emit "Hello World"
2. Terminate

```
/*
 * Good programs have meaningful comments
 */
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

C Program

Hello World

```

/*
 * Good programs have meaningful comments
 */
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}

```

C Program

GNU C Compiler



IA-64 Assembly Language

```

.file "hello.c"
.pred.safe_across_calls p1-p5,p16-p63
.section .rodata.str1.8,"ams",@progbits,1
.align 8
.LC0:
.stringz "Hello world!\n"
.text
.align 16
.global main#
.proc main#
main:
.prologue 12, 33
.save ar.pfs, r34
alloc r34 = ar.pfs, 0, 3, 1, 0
addl r35 = @ltoff(.LC0), gp
.save rp, r33
mov r33 = b0
;;
;body
ld8 r35 = [r35]
br.call.sptk.many b0 = printf#
;;
mov r8 = r0
mov ar.pfs = r34
mov b0 = r33
br.ret.sptk.many b0
.endp main#
.ident "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.2 2.96-112.7.2)"

```

Hello World

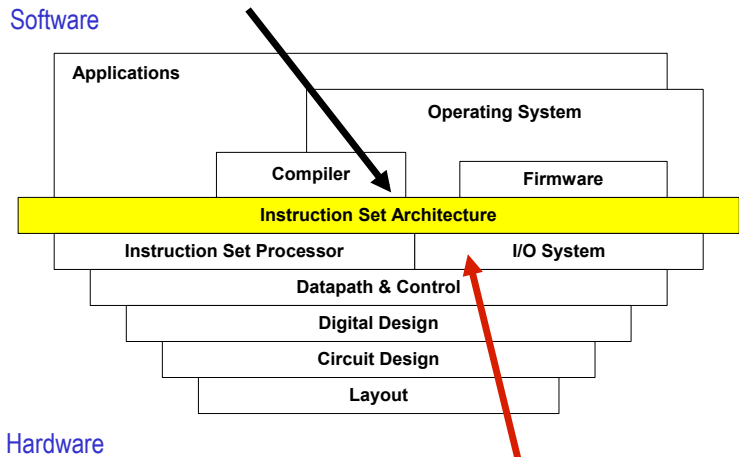
IA-64 Assembly Language

```

.file "hello.c"
.pred.safe_across_calls p1-p5,p16-p63
.section .rodata.str1.8,"ams",@progbits,1
.align 8
.LC0:
.stringz "Hello world!\n"
.text
.align 16
.global main#
.proc main#
main:
.prologue 12, 33
.save ar.pfs, r34
alloc r34 = ar.pfs, 0, 3, 1, 0
addl r35 = @ltoff(.LC0), gp
.save rp, r33
mov r33 = b0
;;
;body
ld8 r35 = [r35]
br.call.sptk.many b0 = printf#
;;
mov r8 = r0
mov ar.pfs = r34
mov b0 = r33
br.ret.sptk.many b0
.endp main#
.ident "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.2 2.96-112.7.2)"

```

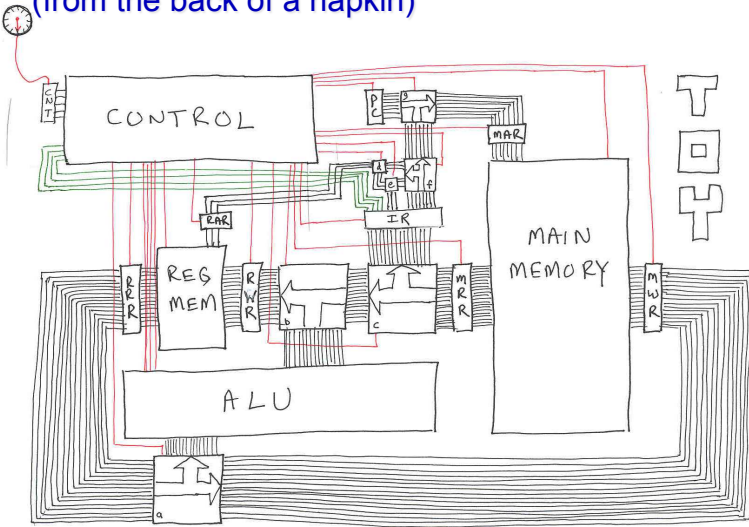
Interfaces in Computer Systems



Hardware

Control

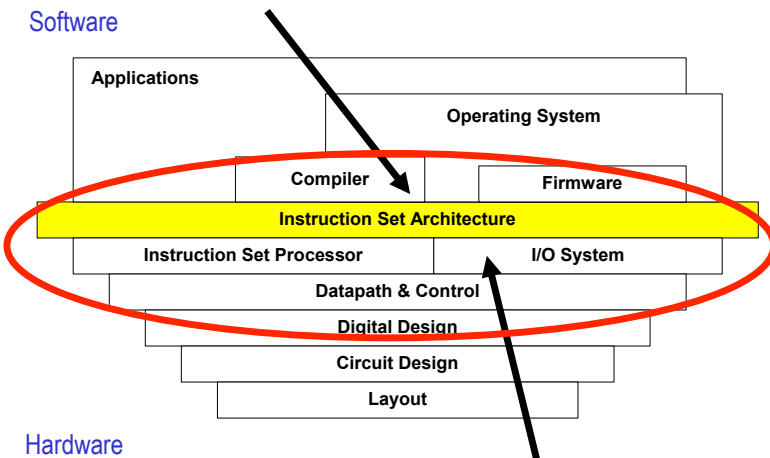
(from the back of a napkin)



61

The Hardware/Software Interface

Software



Hardware

The Instruction Set Architecture

“The vocabulary of commands”

- Defined by the Architecture (x86)
- Implemented by the Machine (Pentium 4, 3.06 GHz)
- An Abstraction Layer: The Hardware/Software Interface
- Architecture has longevity over implementation
- Example:

add r1 = r2 + r3 (assembly)

001 001 010 011 (binary)

Opcode (verb)

Operands (nouns)

Some Figure and Text Acknowledgements

- Dan Connors
- David Patterson
- The COS126 Team (Wayne, Sedgewick)
- A. Mason
- Intel Corporation
- Amazon.com
- www.uiuc.edu