CS 61C: Great Ideas in Computer Architecture *Introduction to Assembly Language and MIPS Instruction Set Architecture*

Instructors:

Bernhard Boser & Randy H. Katz

http://inst.eecs.Berkeley.edu/~cs61c/fa16

Outline

- Assembly Language
- MIPS Architecture
- Registers vs. Variables
- MIPS Instructions
- C-to-MIPS Patterns
- And in Conclusion …

Outline

- Assembly Language
- MIPS Architecture
- Registers vs. Variables
- MIPS Instructions
- C-to-MIPS Patterns
- And in Conclusion …

Levels of Representation/Interpretation

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., MIPS)

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

Anything can be represented as a *number*, i.e., data or instructions

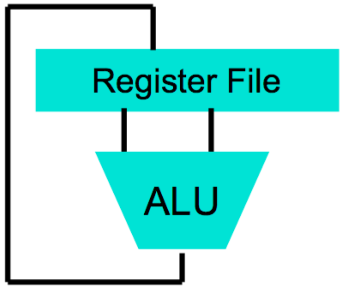
Assembler

Machine Language Program (MIPS)

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

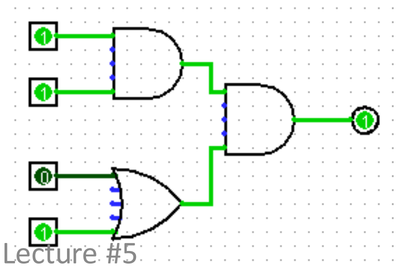
Machine Interpretation

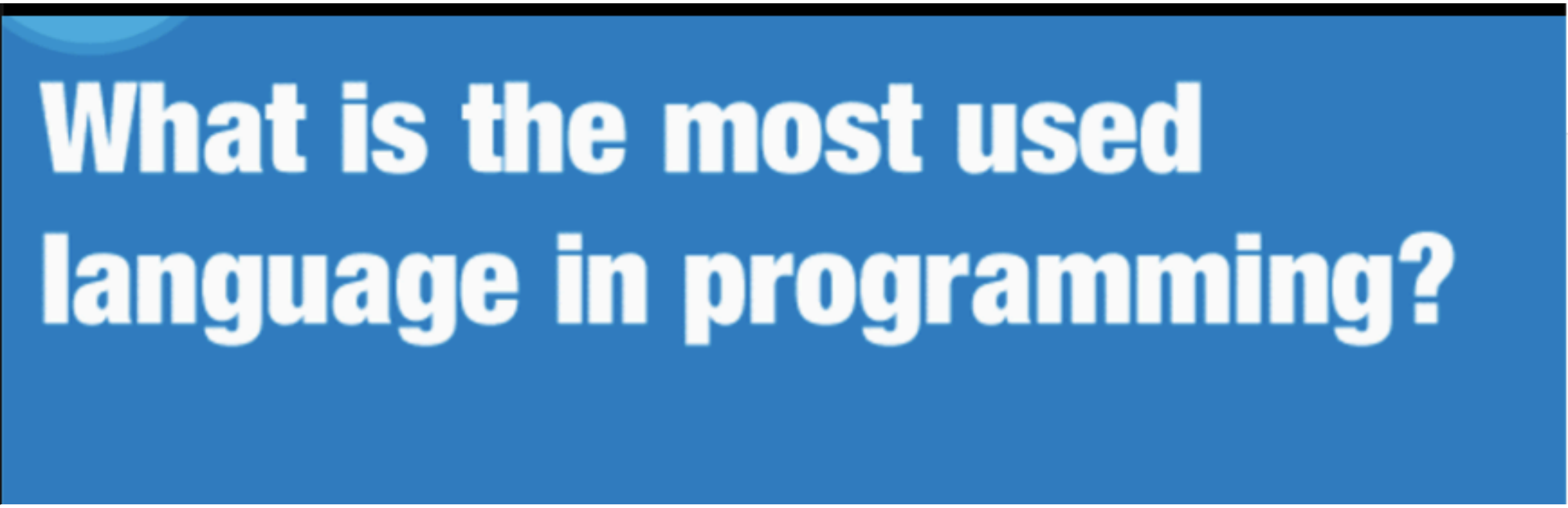
Hardware Architecture Description (e.g., block diagrams)



Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

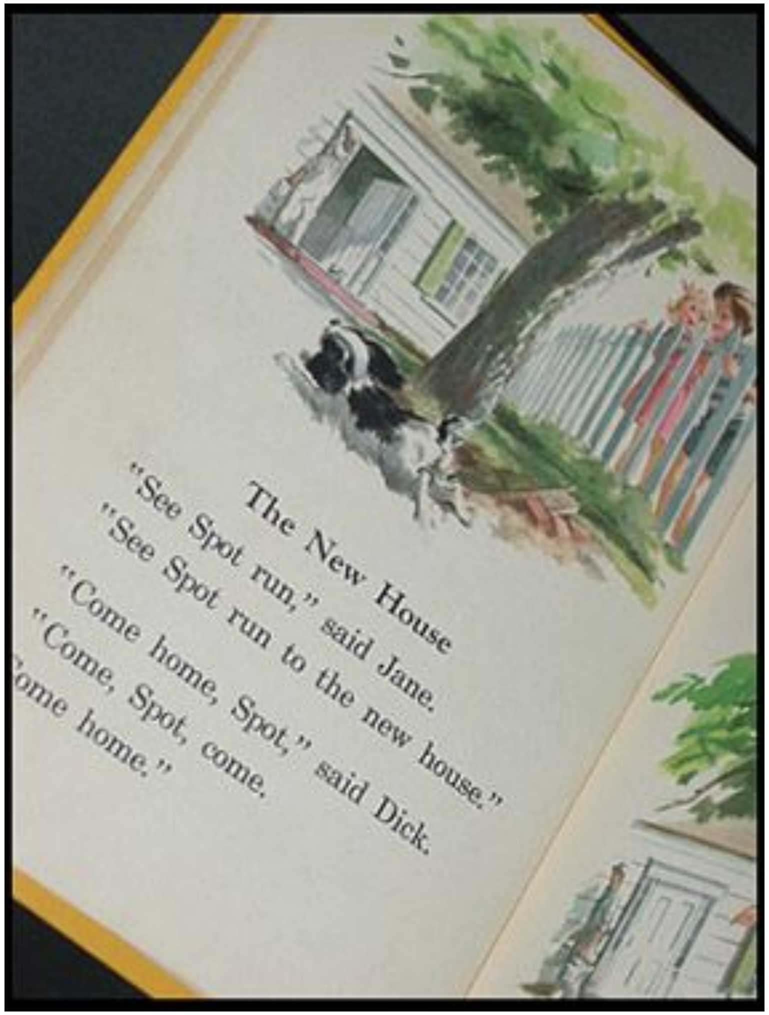




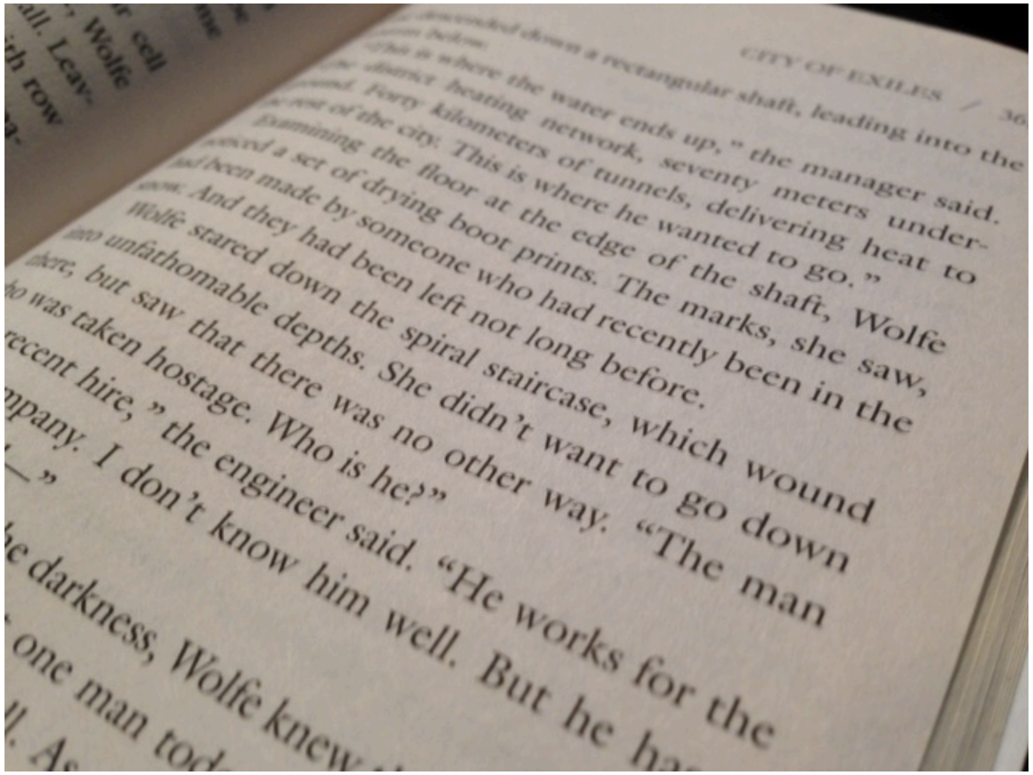
What is the most used language in programming?

Assembly Language

- Job of a CPU (*Central Processing Unit*, aka *Core*): execute *instructions*
- Instructions: CPU's primitives operations
  - Like a sentence: operations (verbs) applied to operands (objects) processed in sequence …
  - With additional operations to change the sequence
- CPUs belong to "families," each implementing its own set of instructions
- CPU's particular set of instructions implements an *Instruction Set Architecture* (*ISA*)
  - Examples: ARM, Intel x86, MIPS, RISC-V, IBM/Motorola PowerPC (old Mac), Intel IA64, ...


The New House

"See Spot run," said Jane.
"See Spot run to the new house."
"Come home, Spot," said Dick.
"Come, Spot, come.
ome home."


r cell
, Wolfe
ll. Leav-
th row

CITY OF EXILES / 36

descended down a rectangular shaft, leading into the
ness below.
"This is where the water ends up," the manager said.
"The district heating network, seventy meters under-
ground. Forty kilometers of tunnels, delivering heat to
the rest of the city. This is where he wanted to go."
Examining the floor at the edge of the shaft, Wolfe
noticed a set of drying boot prints. The marks, she saw,
had been made by someone who had recently been in the
snow. And they had been left not long before.
Wolfe stared down the spiral staircase, which wound
into unfathomable depths. She didn't want to go down
there, but saw that there was no other way. "The man
who was taken hostage. Who is he?"
"recent hire," the engineer said. "He works for the
mpany. I don't know him well. But he ha
—"
e darkness, Wolfe knew
one man tod
ll. As

Assembly Language

High Level Language

Assembly Language

High Level Language

Clicker/Peer Instruction

- For a given function, which programming language likely takes the most lines of code (from most to least)?

  A: Python > MIPS > C

  B: C > Python > MIPS

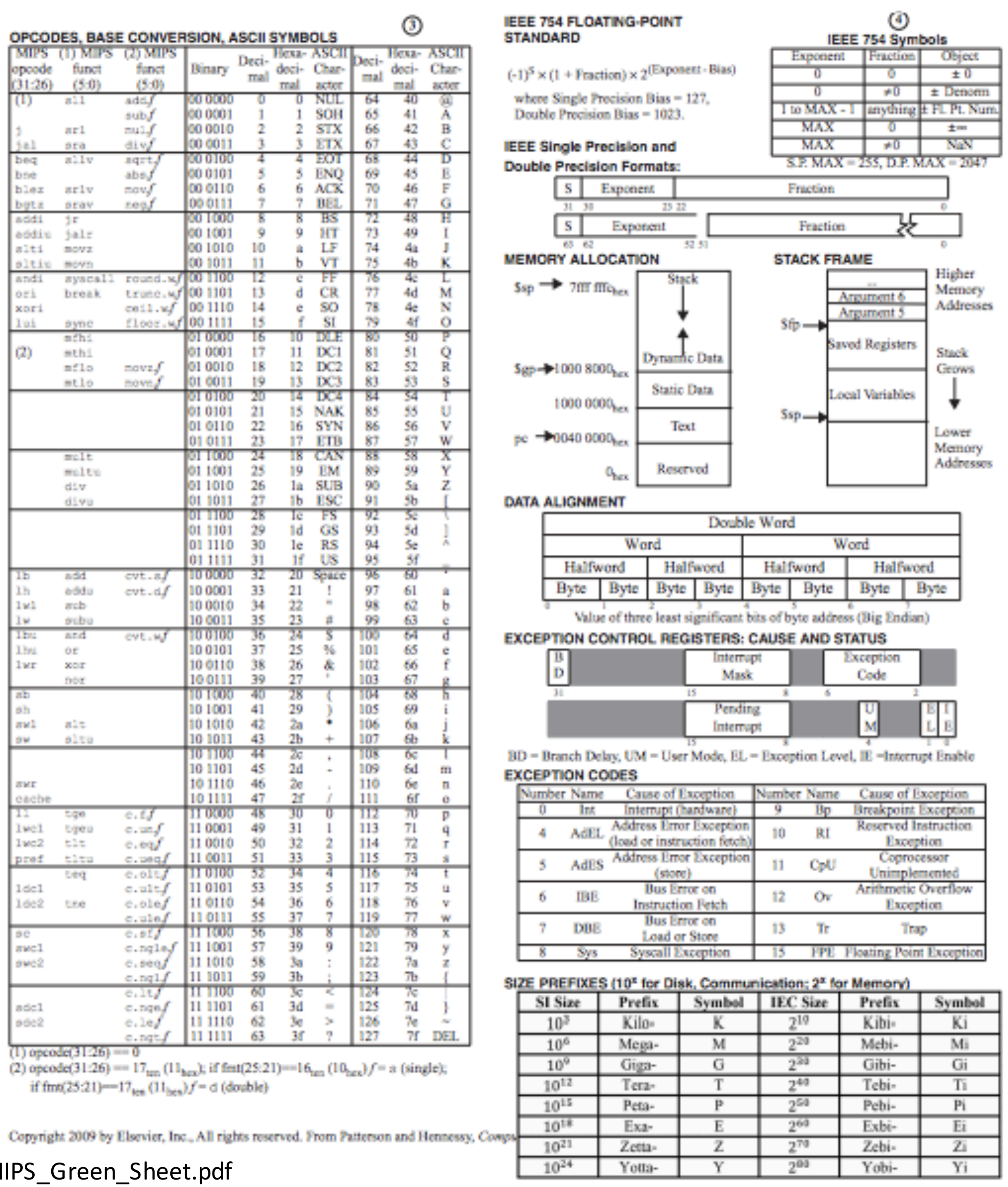
  C: MIPS > Python > C

  [✓ D: MIPS > C > Python]

Instruction **S**et **A**rchitectures

- Early trend: add more instructions to new CPUs for elaborate operations
  - VAX architecture had an instruction to multiply polynomials!
- RISC philosophy (Cocke IBM, Patterson, Hennessy, 1980s) – *Reduced Instruction Set Computing*
  - Keep the instruction set small and simple, in order to build fast hardware
  - Let software do complicated operations by composing simpler ones

MIPS Green Card



(3)

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0
(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single);
if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Compu*

(4)

IEEE 754 FLOATING-POINT STANDARD

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent - Bias})}$

where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

IEEE Single Precision and Double Precision Formats:

S | Exponent | Fraction
31 30 23 22 0

S | Exponent | Fraction
63 62 52 51 0

MEMORY ALLOCATION

$sp → 7fff fffc$_{hex}$ Stack
Dynamic Data
$gp → 1000 8000$_{hex}$ Static Data
1000 0000$_{hex}$ Text
pc → 0040 0000$_{hex}$ Reserved
0$_{hex}$

STACK FRAME

...
Argument 6
Argument 5
$fp → Saved Registers
Local Variables
$sp →

Higher Memory Addresses
Stack Grows
Lower Memory Addresses

DATA ALIGNMENT

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

B D | Interrupt Mask | Exception Code
31 15 8 6 2

Pending Interrupt | U M | E L | I E
15 8 4 1 0

BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

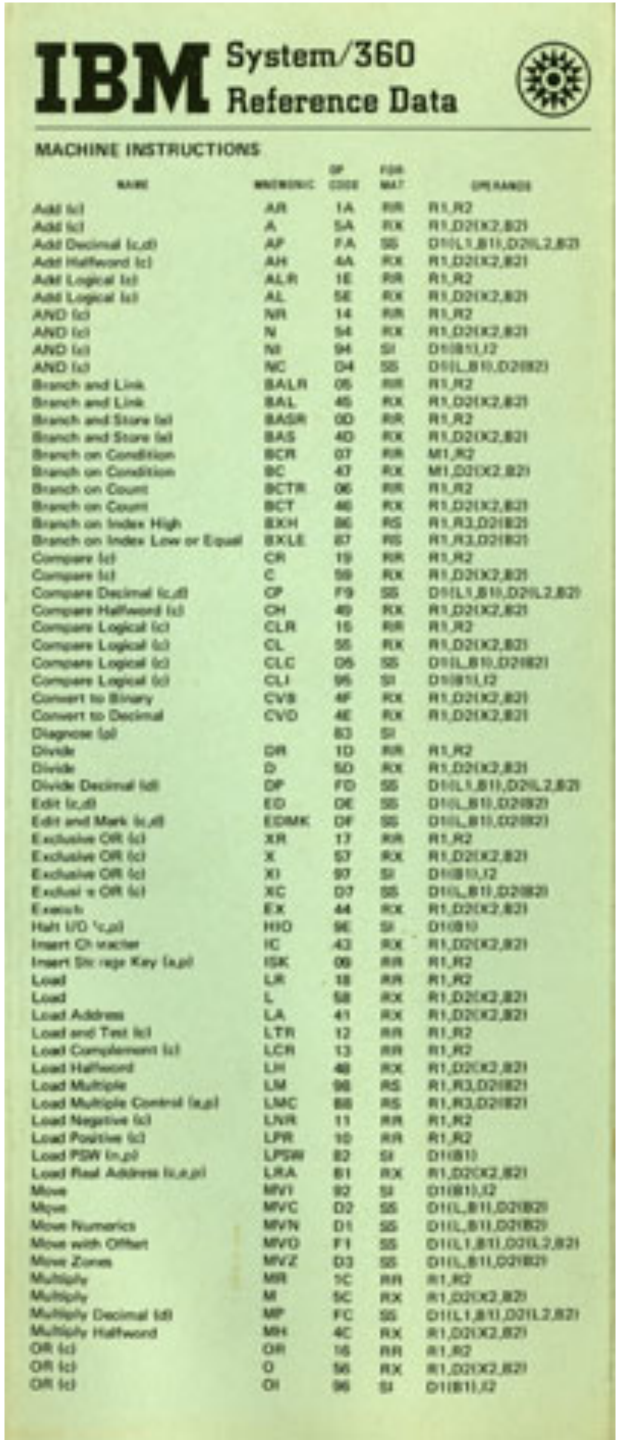
Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SI Size	Prefix	Symbol	IEC Size	Prefix	Symbol
10^3	Kilo-	K	2^{10}	Kibi-	Ki
10^6	Mega-	M	2^{20}	Mebi-	Mi
10^9	Giga-	G	2^{30}	Gibi-	Gi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti
10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi

https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_Green_Sheet.pdf

Inspired by the IBM 360 "Green Card"



IBM System/360 Reference Data

MACHINE INSTRUCTIONS

NAME	MNEMONIC	OP CODE	FOR-MAT	OPERANDS
Add (c)	AR	1A	RR	R1,R2
Add (c)	A	5A	RX	R1,D2(X2,B2)
Add Decimal (c,d)	AP	FA	SS	D1(L1,B1),D2(L2,B2)
Add Halfword (c)	AH	4A	RX	R1,D2(X2,B2)
Add Logical (c)	ALR	1E	RR	R1,R2
Add Logical (c)	AL	5E	RX	R1,D2(X2,B2)
AND (c)	NR	14	RR	R1,R2
AND (c)	N	54	RX	R1,D2(X2,B2)
AND (c)	NI	94	SI	D1(B1),I2
AND (c)	NC	D4	SS	D1(L,B1),D2(B2)
Branch and Link	BALR	05	RR	R1,R2
Branch and Link	BAL	45	RX	R1,D2(X2,B2)
Branch and Store (e)	BASR	0D	RR	R1,R2
Branch and Store (e)	BAS	4D	RX	R1,D2(X2,B2)
Branch on Condition	BCR	07	RR	M1,R2
Branch on Condition	BC	47	RX	M1,D2(X2,B2)
Branch on Count	BCTR	06	RR	R1,R2
Branch on Count	BCT	46	RX	R1,D2(X2,B2)
Branch on Index High	BXH	86	RS	R1,R3,D2(B2)
Branch on Index Low or Equal	BXLE	87	RS	R1,R3,D2(B2)
Compare (c)	CR	19	RR	R1,R2
Compare (c)	C	59	RX	R1,D2(X2,B2)
Compare Decimal (c,d)	CP	F9	SS	D1(L1,B1),D2(L2,B2)
Compare Halfword (c)	CH	49	RX	R1,D2(X2,B2)
Compare Logical (c)	CLR	15	RR	R1,R2
Compare Logical (c)	CL	55	RX	R1,D2(X2,B2)
Compare Logical (c)	CLC	D5	SS	D1(L,B1),D2(B2)
Compare Logical (c)	CLI	95	SI	D1(B1),I2
Convert to Binary	CVB	4F	RX	R1,D2(X2,B2)
Convert to Decimal	CVD	4E	RX	R1,D2(X2,B2)
Diagnose (p)		83	SI	
Divide	DR	1D	RR	R1,R2
Divide	D	5D	RX	R1,D2(X2,B2)
Divide Decimal (d)	DP	FD	SS	D1(L1,B1),D2(L2,B2)
Edit (c,d)	ED	DE	SS	D1(L,B1),D2(B2)
Edit and Mark (c,d)	EDMK	DF	SS	D1(L,B1),D2(B2)
Exclusive OR (c)	XR	17	RR	R1,R2
Exclusive OR (c)	X	57	RX	R1,D2(X2,B2)
Exclusive OR (c)	XI	97	SI	D1(B1),I2
Exclusive OR (c)	XC	D7	SS	D1(L,B1),D2(B2)
Execute	EX	44	RX	R1,D2(X2,B2)
Halt I/O (c,p)	HIO	9E	SI	D1(B1)
Insert Character	IC	43	RX	R1,D2(X2,B2)
Insert Storage Key (a,p)	ISK	09	RR	R1,R2
Load	LR	18	RR	R1,R2
Load	L	58	RX	R1,D2(X2,B2)
Load Address	LA	41	RX	R1,D2(X2,B2)
Load and Test (c)	LTR	12	RR	R1,R2
Load Complement (c)	LCR	13	RR	R1,R2
Load Halfword	LH	48	RX	R1,D2(X2,B2)
Load Multiple	LM	98	RS	R1,R3,D2(B2)
Load Multiple Control (e,p)	LMC	B8	RS	R1,R3,D2(B2)
Load Negative (c)	LNR	11	RR	R1,R2
Load Positive (c)	LPR	10	RR	R1,R2
Load PSW (n,p)	LPSW	82	SI	D1(B1)
Load Real Address (c,e,p)	LRA	B1	RX	R1,D2(X2,B2)
Move	MVI	92	SI	D1(B1),I2
Move	MVC	D2	SS	D1(L,B1),D2(B2)
Move Numerics	MVN	D1	SS	D1(L,B1),D2(B2)
Move with Offset	MVO	F1	SS	D1(L1,B1),D2(L2,B2)
Move Zones	MVZ	D3	SS	D1(L,B1),D2(B2)
Multiply	MR	1C	RR	R1,R2
Multiply	M	5C	RX	R1,D2(X2,B2)
Multiply Decimal (d)	MP	FC	SS	D1(L1,B1),D2(L2,B2)
Multiply Halfword	MH	4C	RX	R1,D2(X2,B2)
OR (c)	OR	16	RR	R1,R2
OR (c)	O	56	RX	R1,D2(X2,B2)
OR (c)	OI	96	SI	D1(B1),I2

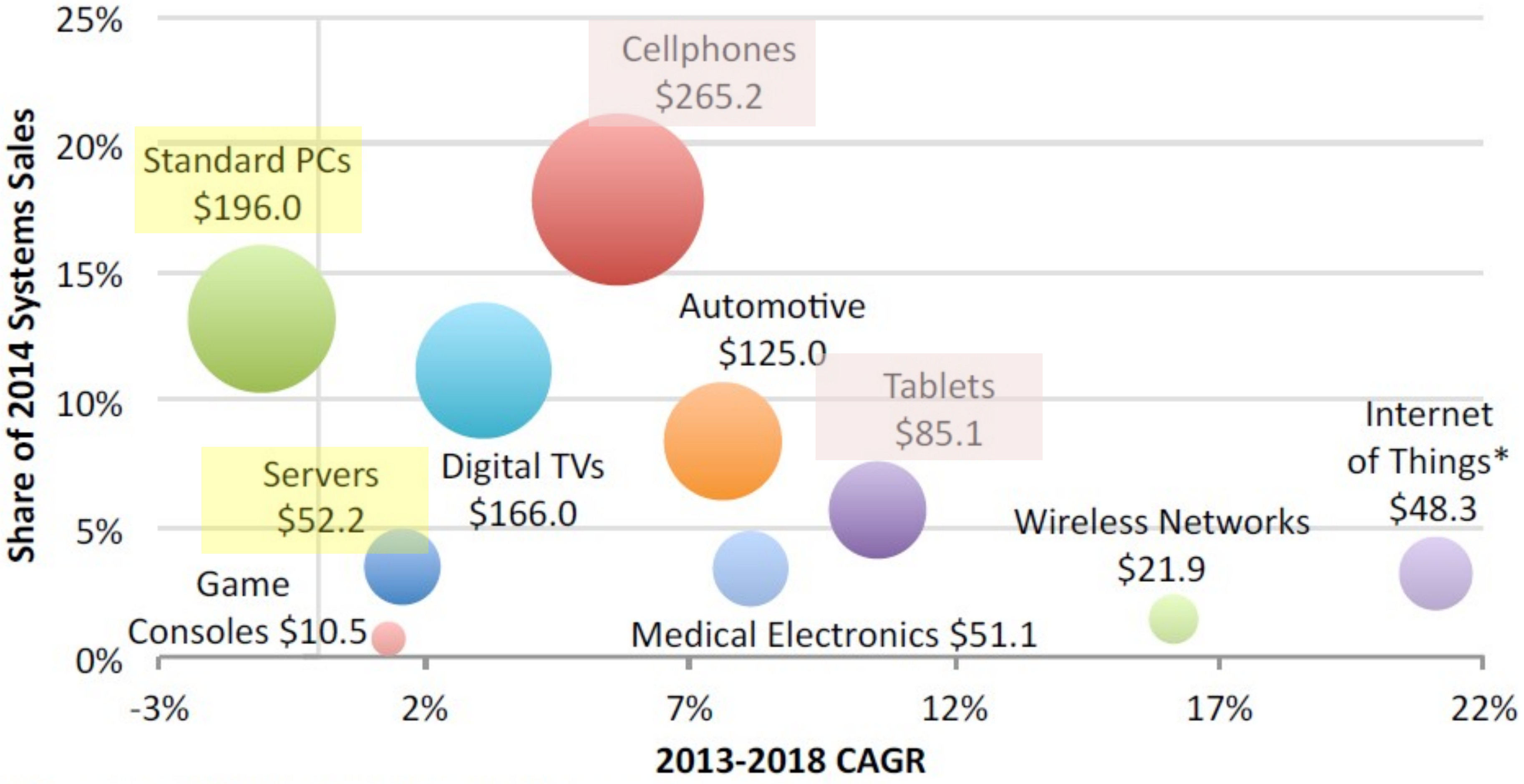
Outline

- Assembly Language
- MIPS Architecture
- Registers vs. Variables
- MIPS Instructions
- C-to-MIPS Patterns
- And in Conclusion …

MIPS Architecture

- MIPS: semiconductor company that built one of the first commercial RISC architectures (1984-2013, acquired by Imagination Technologies)
- Why MIPS instead of Intel x86 (or ARM)?
  - MIPS is simple, elegant; avoid getting bogged down in gritty details
  - MIPS (used to be) widely used in embedded apps, e.g., consumer electronics and network routers; x86 little used in embedded and lots more embedded computers than PCs
  - Nevertheless, cs61c migrating to ARM next semester!

End-Use Systems Markets ($B) and Growth Rates



*Covers only the Internet connection portion of systems

Source: IC Insights

Number One in Digital Home CPUs

MIPS TECHNOLOGIES

Number One Market Share

Digital TV

Cable, Satellite & IPTV Set-top Boxes

Blu-ray Disc Players

DVD; DVR

Digital Cameras

Broadband CPE

WiFi Access Points and Routers

*IDC Research, 2008 embedded processor share

Broadcom, Microchip, Altera, Sony, Maxim, MStar, Cisco, ST, Motorola, Novatek, PMC-Sierra, NetLogic, LSI, Cpulan, Sigma, ALi, Zoran, Atheros, Sunplus, Actions, Cavium Networks, Realtek, Haier, MegaChips, Toshiba, ViXS, ICT, NEC, Trident, Infineon, Socle, NXP, pixelworks, Marvell, Ralink, SiS, Entropic communications, albeir, AMIMON, Percello, MobilEye, Magnum, MetaLink Broadband, IDT, Centillium Communications, K-micro, BroadLight, Mavrix Technology, Semtek, Magic Pixel Inc., Beceem, TranSwitch, Huaya Micro

Assembly Variables: Registers

- Unlike HLL like C or Java, assembly does not have *variables* as you know and love them
  - More primitive, closer what simple hardware can directly support
- Assembly operands are objects called registers
  - Limited number of special places to hold values, built directly into the hardware
  - Operations can only be performed on these!
- Benefit: Since registers are directly in hardware, they are very fast (faster than 1 ns - light travels 1 foot in 1 ns!!!)

Outline

- Assembly Language
- MIPS Architecture
- Registers vs. Variables
- MIPS Instructions
- C-to-MIPS Patterns
- And in Conclusion …

Number of MIPS Registers

- Drawback: Since registers are in hardware, there are a limited number of them
  - Solution: MIPS code must be carefully written to to efficiently use registers
- 32 registers in MIPS
  - Why 32? Smaller is faster, but too small is bad. Goldilocks principle ("This porridge is too hot; This porridge is too cold; this porridge is just right")
- Each MIPS register is 32 bits wide
  - Groups of 32 bits called a word in MIPS ISA

Names of MIPS Registers

- Registers are numbered from 0 to 31
- Each register can be referred to by number or name
- Number references:
  - $0, $1, $2, … $30, $31
- For now:
  - $16 - $23 ➔ $s0 - $s7 (can hold things like C variables)
  - $8 - $15 ➔ $t0 - $t7 (can hold temporary variables)
  - Later will explain other 16 register names
- In general, use names to make your code more readable

C, Java Variables vs. Registers

- In C (and most HLLs):
  - Variables declared and given a type
    - Example: `int fahr, celsius;`
      `char a, b, c, d, e;`
  - Each variable can ONLY represent a value of the type it was declared (e.g., cannot mix and match *int* and *char* variables)
- In Assembly Language:
  - Registers have no type;
  - Operation determines how register contents are interpreted

Outline

- Assembly Language
- MIPS Architecture
- Registers vs. Variables
- MIPS Instructions
- C-to-MIPS Patterns
- And in Conclusion …

Addition and Subtraction of Integers

- Addition in Assembly
  - Example: `add $s0,$s1,$s2` (in MIPS)
  - Equivalent to: a = b + c (in C)

  where C variables ⇔ MIPS registers are:

  a ⇔ $s0, b ⇔ $s1, c ⇔ $s2
- Subtraction in Assembly
  - Example: `sub $s3,$s4,$s5` (in MIPS)
  - Equivalent to: d = e - f (in C)

  where C variables ⇔ MIPS registers are:
  d ⇔ $s3, e ⇔ $s4, f ⇔ $s5

Addition and Subtraction of Integers Example 1

- How to do the following C statement?

  a = b + c + d - e;

- Break into multiple instructions

```
add $t0, $s1, $s2 # temp = b + c
add $t0, $t0, $s3 # temp = temp + d
sub $s0, $t0, $s4 # a = temp - e
```

- A single line of C may break up into several lines of MIPS
- Notice the use of temporary registers – don't want to modify the variable registers `$s`
- Everything after the hash mark on each line is ignored (comments)

Immediates

- Immediates are numerical constants
- They appear often in code, so there are special instructions for them
- Add Immediate:

```
    addi $s0,$s1,-10   (in MIPS)
         f = g - 10    (in C)
```

  where MIPS registers `$s0,$s1` are associated with C variables **f, g**

- Syntax similar to `add` instruction, except that last argument is a number instead of a register

```
    add $s0,$s1,$zero  (in MIPS)
        f = g          (in C)
```

Overflow in Arithmetic

- Reminder: Overflow occurs when there is an error in arithmetic due to the limited precision in computers
- Example (4-bit unsigned numbers):

```
   15            1111
+   3         +  0011
-----         -------
   18           10010
```

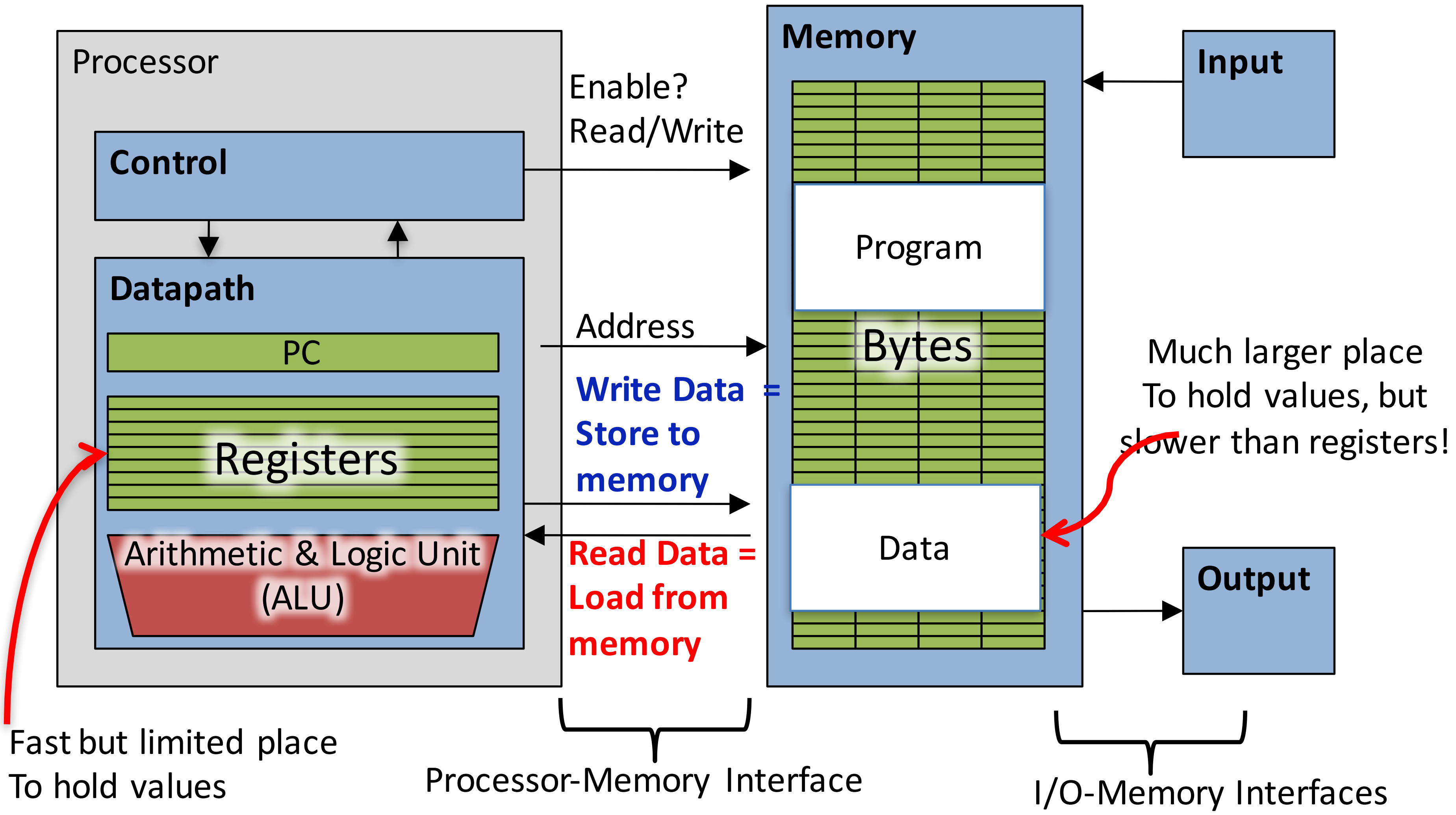
- But we don't have room for 5-bit solution, so the solution would be `0010`, which is +2, and "wrong"

Overflow handling in MIPS

- Some languages detect overflow (Ada), some don't (most C implementations)
- MIPS solution is two alternative arithmetic instructions:
  - Cause overflow to be detected (e.g., calculations):
    - add (add)
    - add immediate (addi)
    - subtract (sub)
  - Don't cause overflow detection (e.g., pointer arithmetic)
    - add unsigned (addu)
    - add immediate unsigned (addiu)
    - subtract unsigned (subu)
- Compiler selects appropriate arithmetic
  - MIPS C compilers produce addu, addiu, subu

Break!

Data Transfer: Load from and Store to memory


Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Enable?
Read/Write
Address
Write Data = Store to memory
Read Data = Load from memory
Memory
Program
Bytes
Data
Input
Output
Much larger place To hold values, but slower than registers!
Fast but limited place To hold values
Processor-Memory Interface
I/O-Memory Interfaces

Memory Addresses are in Bytes

- Data typically smaller than 32 bits, but rarely smaller than 8 bits (e.g., char type)–works fine if everything is a multiple of 8 bits
- 8 bit chunk is called a *byte* (1 word = 4 bytes)
- Memory addresses are really in *bytes*, not words
- Word addresses are 4 bytes apart
  – Word address is same as address of leftmost byte – most significant byte (i.e. Big-endian convention)

Most significant byte in a word

↓

...	...	...	...
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3
31 24	23 16	15 8	7 0

Most significant byte gets the smallest address

Transfer from Memory to Register

- C code

```
int  A[100];
g = h + A[3];
```

- Using Load Word (`lw`) in MIPS:

```
lw   $t0,12($s3)   # Temp reg $t0 gets A[3]
add $s1,$s2,$t0    # g = h + A[3]
```

Note: `$s3` – base register (pointer)
12 – offset in bytes
Offset must be a constant known at assembly time

Transfer from Register to Memory

- C code

```
int  A[100];
A[10] = h + A[3];
```

- Using Store Word (sw) in MIPS:

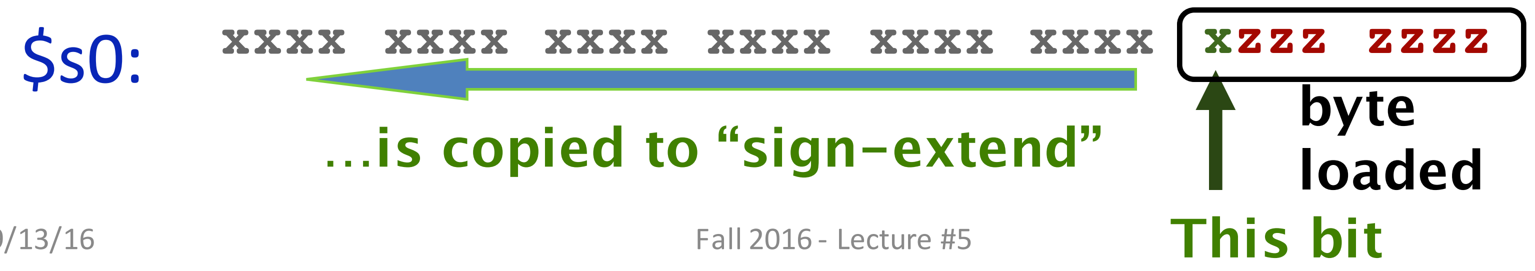
```
lw   $t0,12($s3)     # Temp reg $t0 gets A[3]
add  $t0,$s2,$t0     # Temp reg $t0 gets h + A[3]
sw   $t0,40($s3)     # A[10] = h + A[3]
```

Note: $s3 – base register (pointer)
12,40 – offsets in bytes

$s3+12 and $s3+40 must be multiples of 4

Loading and Storing Bytes

- In addition to word data transfers (`lw`, `sw`), MIPS has byte data transfers:
  - load byte: `lb`
  - store byte: `sb`
- Same format as `lw`, `sw`
- E.g., `lb $s0,3($s1)`
  - contents of memory location with address = sum of "3" + contents of register *$s1* is copied to the low byte position of register *$s0*.



Speed of Registers vs. Memory

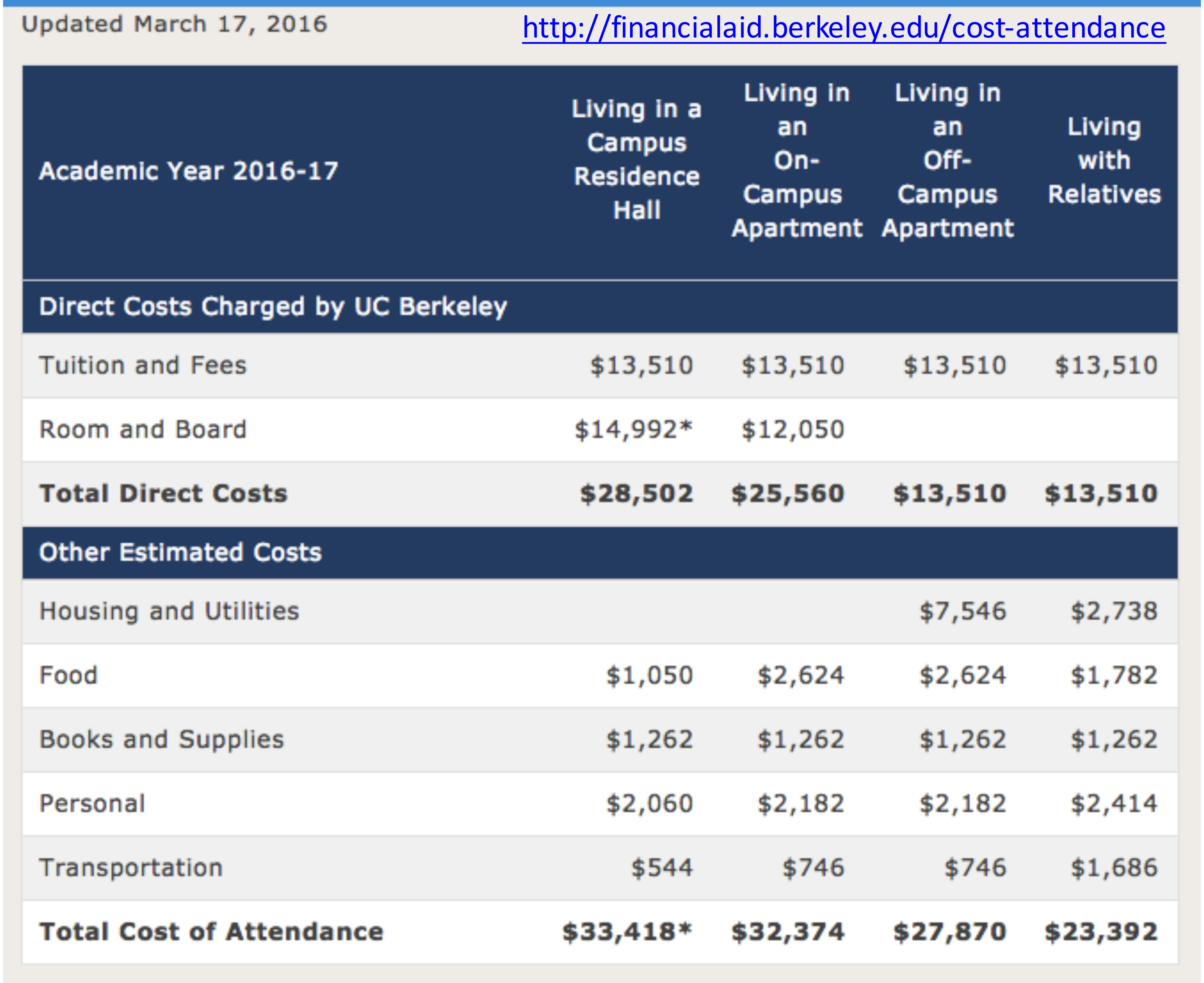
- Given that
  - Registers: 32 words (128 Bytes)
  - Memory: Billions of bytes (2 GB to 8 GB on laptop)
- and the RISC principle is…
  - Smaller is faster
- How much faster are registers than memory??
- About 100-500 times faster!
  - in terms of *latency* of one access

Administrivia

- HW #0 due tonight!
- Lab #1, Project #1 published (soon)
- Guerrilla Review sessions to start soon, possibly next week
  - C practice
- Three weeks to Midterm #1!
  - We have started working on it.

Laptops, Revisted

http://financialaid.berkeley.edu/cost-attendance

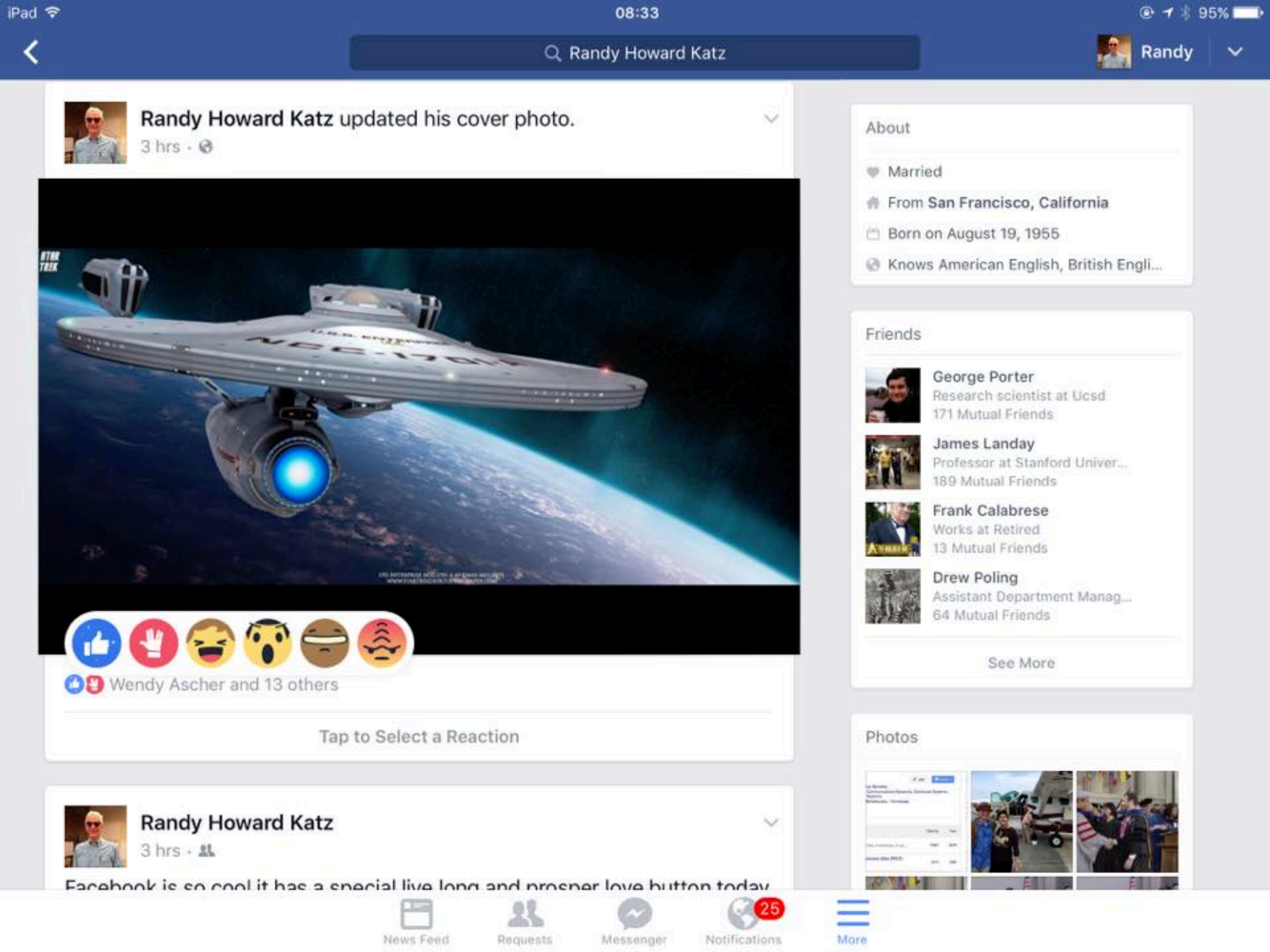


Updated March 17, 2016

Academic Year 2016-17	Living in a Campus Residence Hall	Living in an On-Campus Apartment	Living in an Off-Campus Apartment	Living with Relatives
Direct Costs Charged by UC Berkeley				
Tuition and Fees	$13,510	$13,510	$13,510	$13,510
Room and Board	$14,992*	$12,050		
Total Direct Costs	**$28,502**	**$25,560**	**$13,510**	**$13,510**
Other Estimated Costs				
Housing and Utilities			$7,546	$2,738
Food	$1,050	$2,624	$2,624	$1,782
Books and Supplies	$1,262	$1,262	$1,262	$1,262
Personal	$2,060	$2,182	$2,182	$2,414
Transportation	$544	$746	$746	$1,686
Total Cost of Attendance	**$33,418***	**$32,374**	**$27,870**	**$23,392**

Last Five Minutes, Please!

ARRRGH - NOISY CHAIRS DRIVING YOU CRAZY?
The answer is simple .....
ChairGlide
Noisy chair stoppers
A Great British Invention!



iPad 08:33 95%

Randy Howard Katz

Randy

Randy Howard Katz updated his cover photo.

3 hrs ·

Wendy Ascher and 13 others

Tap to Select a Reaction

Randy Howard Katz

3 hrs ·

Facebook is so cool it has a special live long and prosper love button today

About

Married

From San Francisco, California

Born on August 19, 1955

Knows American English, British Engli...

Friends

George Porter
Research scientist at Ucsd
171 Mutual Friends

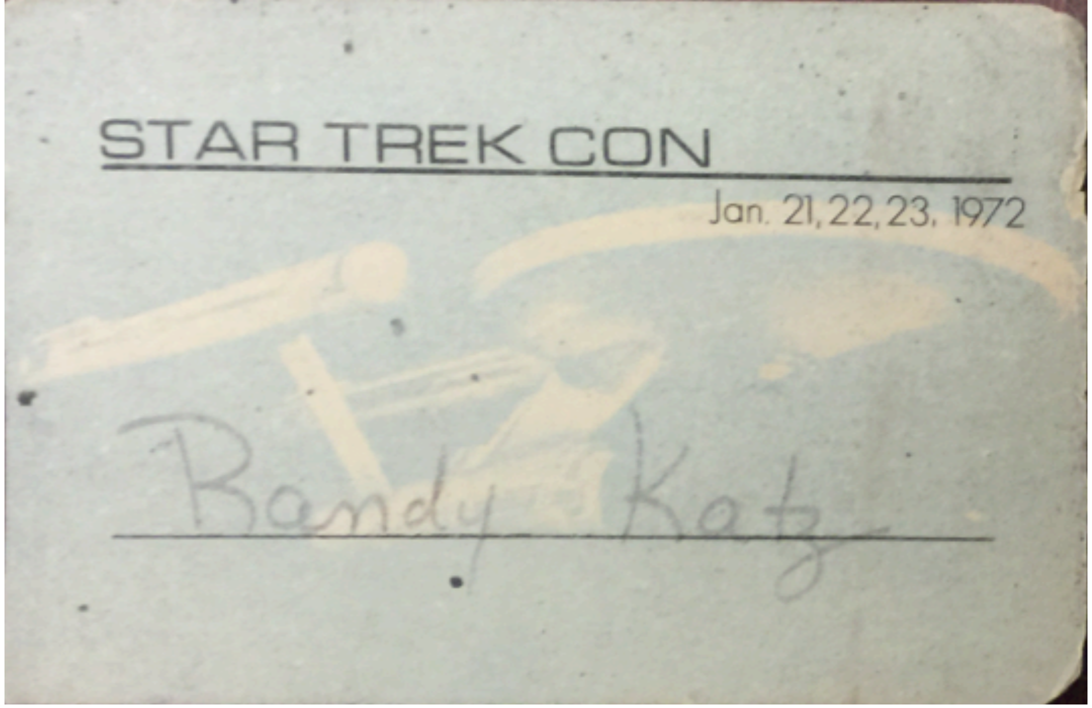
James Landay
Professor at Stanford Univer...
189 Mutual Friends

Frank Calabrese
Works at Retired
13 Mutual Friends

Drew Poling
Assistant Department Manag...
64 Mutual Friends

See More

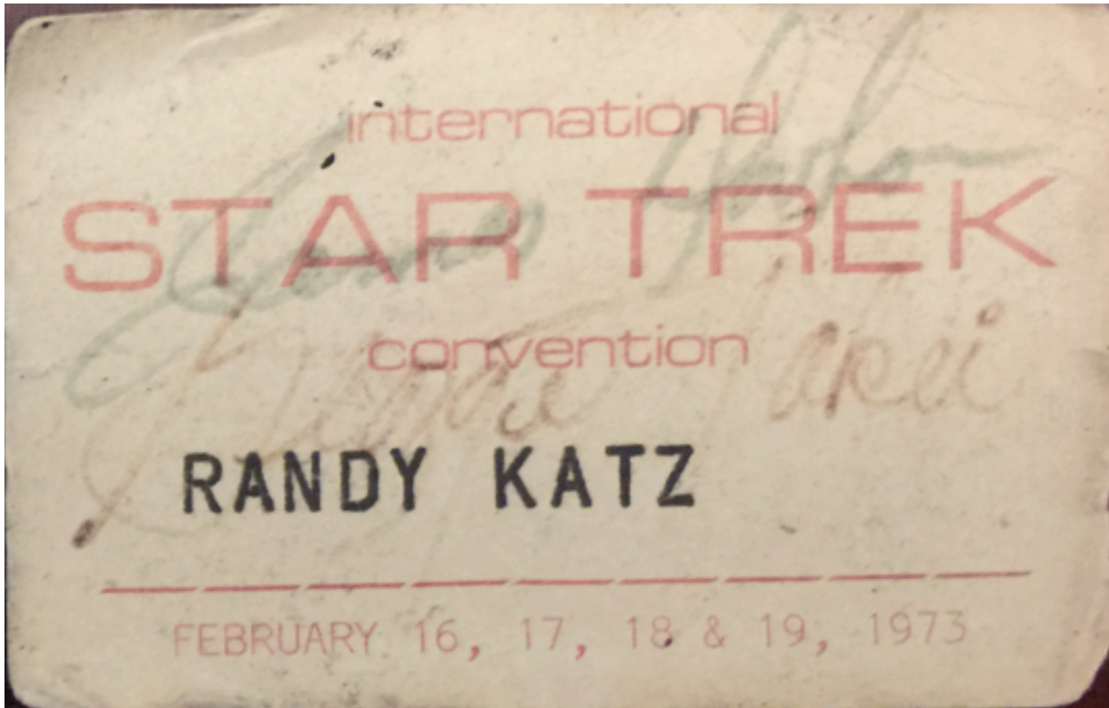
Photos



STAR TREK CON

Jan. 21, 22, 23, 1972

Randy Katz



international

STAR TREK

convention

[illegible]

RANDY KATZ

FEBRUARY 16, 17, 18 & 19, 1973

Break!

Outline

- Assembly Language
- MIPS Architecture
- Registers vs. Variables
- MIPS Instructions
- C-to-MIPS Patterns
- And in Conclusion …

MIPS Logical Instructions

- Useful to operate on fields of bits within a word
  - e.g., characters within a word (8 bits)
- Operations to pack /unpack bits into words
- Called *logical operations*

Logical operations	C operators	Java operators	MIPS instructions
Bit-by-bit AND	&	&	**`and`**
Bit-by-bit OR	|	|	**`or`**
Bit-by-bit NOT	~	~	**`not`**
Shift left	<<	<<	**`sll`**
Shift right	>>	>>	**`srl`**

Logic Shifting

- Shift Left: `sll $s1,$s2,2` #s1=s2<<2
  - Store in $s1 the value from $s2 shifted 2 bits to the left (they fall off end), inserting 0's on right; << in C

  Before: $0000\ 0002_{hex}$

  $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$

  After: $0000\ 000\underline{8}_{hex}$

  $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 10\underline{00}_{two}$

What arithmetic effect does shift left have?

- Shift Right: `srl` is opposite shift; >>

Arithmetic Shifting

- Shift right arithmetic moves *n* bits to the right (insert high order sign bit into empty bits)
- For example, if register $s0 contained

  1111 1111 1111 1111 1111 1111 1110 0111$_{two}$= -25$_{ten}$
- If executed sra $s0, $s0, 4, result is:

  1111 1111 1111 1111 1111 1111 1111 1110$_{two}$= -2$_{ten}$
- Unfortunately, this is NOT same as dividing by 2^n
  - Fails for odd negative numbers
  - C arithmetic semantics is that division should round towards 0

Computer Decision Making

- Based on computation, do something different
- In programming languages: *if*-statement

- MIPS: *if*-statement instruction is

  ```
  beq register1,register2,L1
  ```

  means: go to statement labeled L1
  if (value in register1) == (value in register2)
  ….otherwise, go to next statement
- `beq` stands for *branch if equal*
- Other instruction: `bne` for *branch if not equal*

Types of Branches

- **Branch** – change of control flow
- **Conditional Branch** – change control flow depending on outcome of comparison
  - branch *if* equal (`beq`) or branch *if not* equal (`bne`)
- **Unconditional Branch** – always branch
  - a MIPS instruction for this*: jump (*`j`*)*

Example *if* Statement

- Assuming translations below, compile *if* block

  f → `$s0`　g → `$s1`　h → `$s2`

  i → `$s3`　j → `$s4`

```
if (i == j)          bne $s3,$s4,Exit
  f = g + h;         add $s0,$s1,$s2
               Exit:
```

- May need to negate branch condition

Example *if-else* Statement

- Assuming translations below, compile

  f → `$s0` g → `$s1` h → `$s2`

  i → `$s3` j → `$s4`

```
if (i == j)              bne $s3,$s4,Else
  f = g + h;             add $s0,$s1,$s2
else                     j Exit
  f = g - h;       Else: sub $s0,$s1,$s2
                   Exit:
```

Inequalities in MIPS

- Until now, we've only tested equalities (== and != in C); General programs need to test < and > as well.
- Introduce MIPS Inequality Instruction: "Set on Less Than"

  Syntax: `slt reg1,reg2,reg3`

  Meaning:
  ```
  if (reg2 < reg3)
      reg1 = 1;
  else reg1 = 0;
  ```

  "set" means "change to 1",
  "reset" means "change to 0".

Inequalities in MIPS (cont)

- How do we use this? Compile by hand:
  if (g < h) goto Less;   #g:$s0, h:$s1
- Answer: compiled MIPS code…
  # $t0 = 1 if g<h
  # if $t0!=0 goto Less
- Register $zero always contains the value 0, so bne and beq often use it for comparison after an slt instruction
- `sltu` treats registers as unsigned

Inequalities in MIPS (cont)

- How do we use this? Compile by hand:

```
if (g < h) goto Less;        #g:$s0, h:$s1
```

- Answer: compiled MIPS code…

```
slt $t0,$s0,$s1          # $t0 = 1 if g<h
bne $t0,$zero,Less       # if $t0!=0 goto Less
```

- Register $zero always contains the value 0, so bne and beq often use it for comparison after an slt instruction
- `sltu` treats registers as unsigned

Immediates in Inequalities

- `slti` an immediate version of `slt` to test against constants

```
Loop:   . . .

slti $t0,$s0,1             # $t0 = 1 if
                           # $s0<1
beq  $t0,$zero,Loop        # goto Loop
                           # if $t0==0
                           # (if ($s0>=1))
```

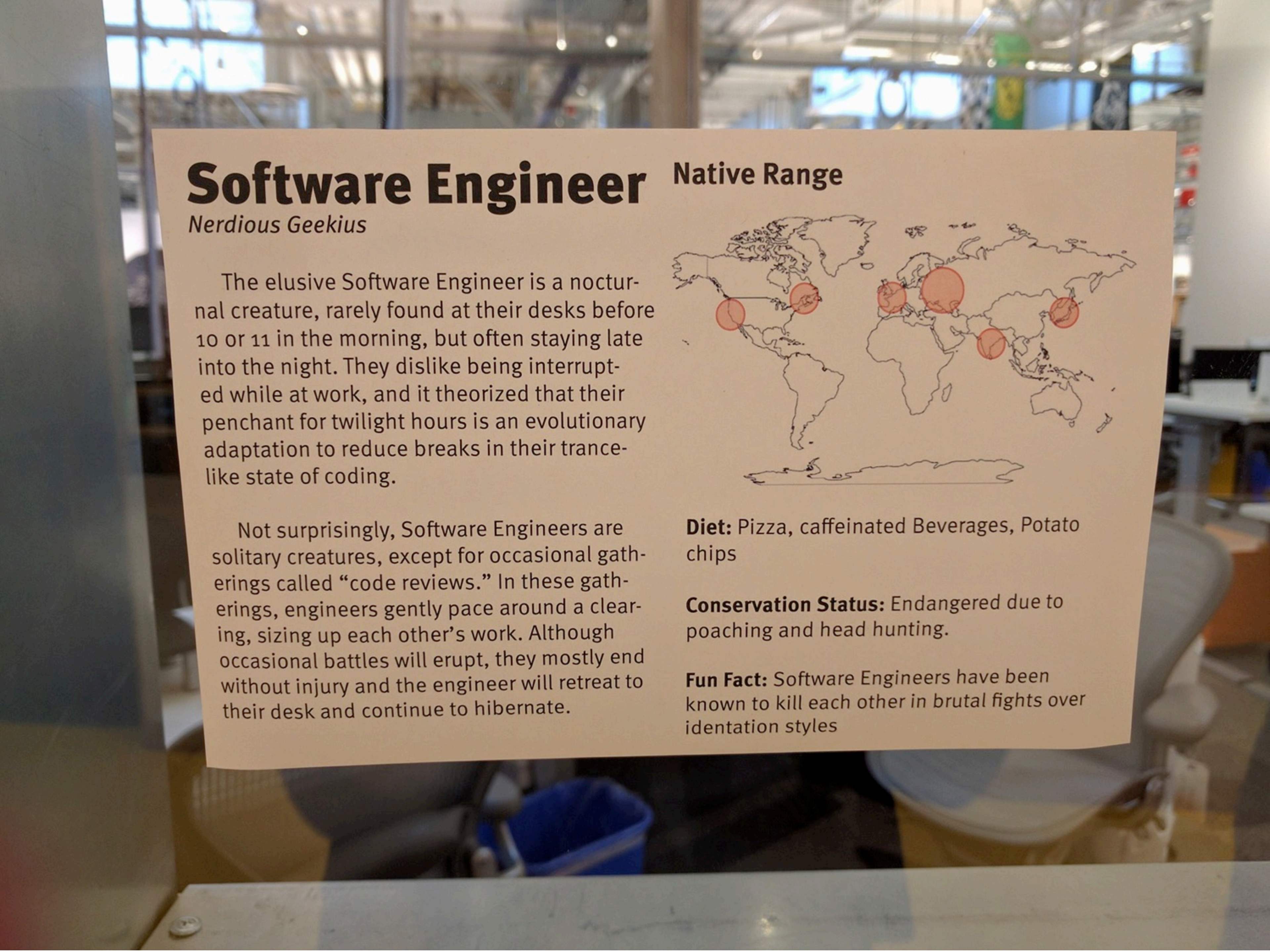
Loops in C/Assembly

- Simple loop in C; A[] is an array of ints

```
do { g = g + A[i];
         i = i + j;
} while (i != h);
```

- Use this mapping: g, h, i, j, &A[0]
  $s1, $s2, $s3, $s4, $s5

```
Loop:                         # $t1= 4*i
                              # $t1=addr A+4i
                              # $t1=A[i]
                              # g=g+A[i]
                           # i=i+j
                              # goto Loop
                           # if i!=h
```



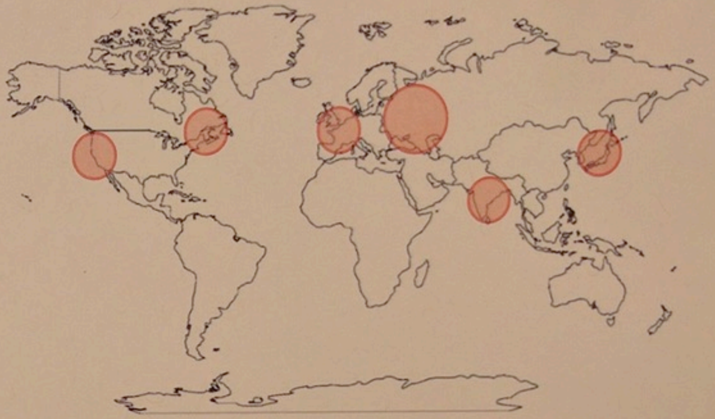
Software Engineer

Nerdious Geekius

The elusive Software Engineer is a nocturnal creature, rarely found at their desks before 10 or 11 in the morning, but often staying late into the night. They dislike being interrupted while at work, and it theorized that their penchant for twilight hours is an evolutionary adaptation to reduce breaks in their trance-like state of coding.

Not surprisingly, Software Engineers are solitary creatures, except for occasional gatherings called “code reviews.” In these gatherings, engineers gently pace around a clearing, sizing up each other’s work. Although occasional battles will erupt, they mostly end without injury and the engineer will retreat to their desk and continue to hibernate.

Native Range



Diet: Pizza, caffeinated Beverages, Potato chips

Conservation Status: Endangered due to poaching and head hunting.

Fun Fact: Software Engineers have been known to kill each other in brutal fights over identation styles

And In Conclusion

BBC AMERICA
STAR TREK: 50TH ANNIVERSARY MARATHON
Uncut. Remastered. High Definition.
STARTING THURSDAY AT 8:30/7:30C ››