Final Exam Review

COMP 273 – Fall 2021

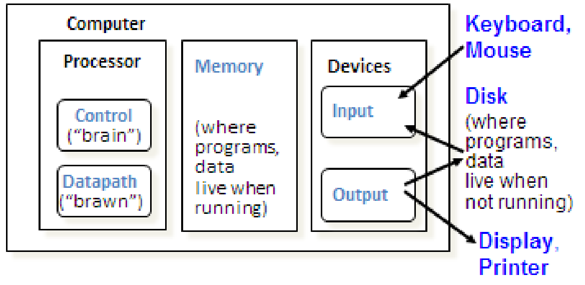
Slide deck number, roughly corresponding to lectures…

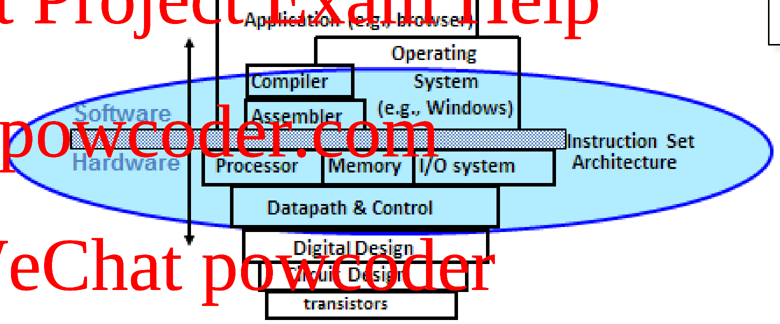
Introduction to Machine Structures

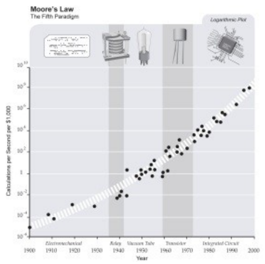
L1, PH 1.1-1.3

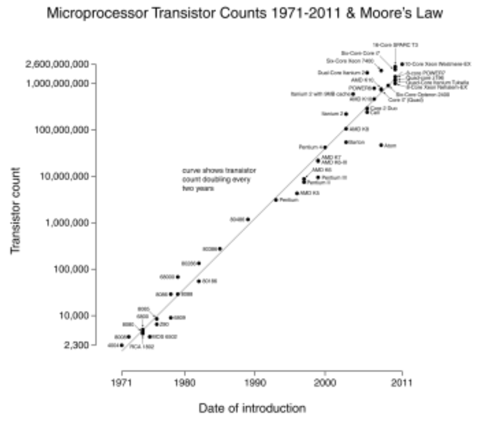
Textbook (Patterson and Hennessey) sections (see end of these slides wrt edition numbers)

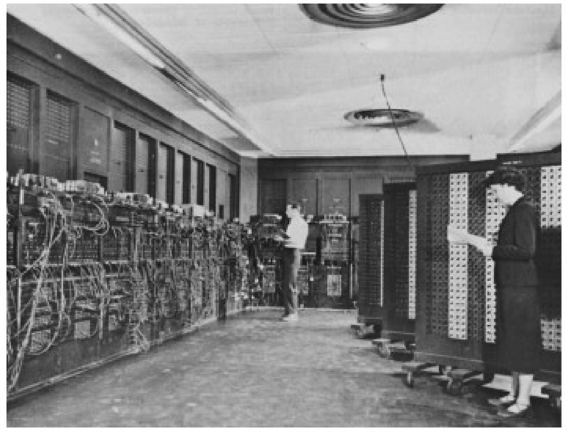
- The **5 components** of a PC
  - Control + Datapath (the processor)
  - Memory
  - Input and Output devices
- The Big Picture
  - High Level Language to Assembly Language (the compiler)
  - Assembly Language to machine code (the assembler)
- Technology trends







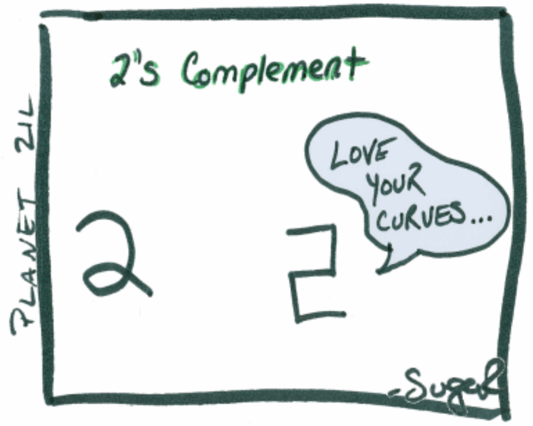


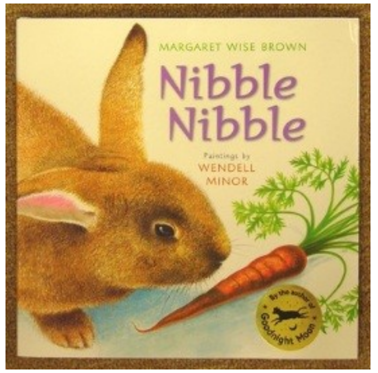


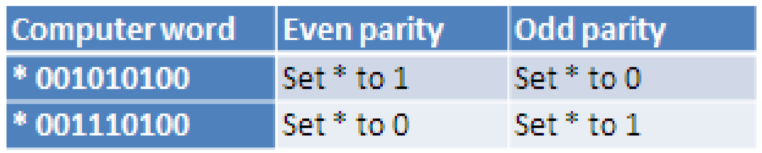
Number Representation

L2, Provided Notes, PH 2.4, 3.1-3.4

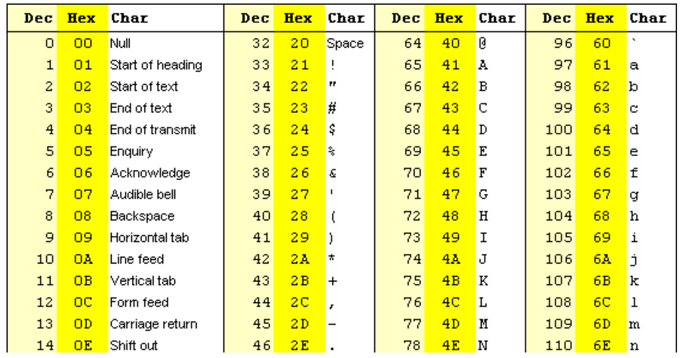
- Positional notation
- Conversion between bases
  10 (decimal), 2 (binary),
  8 (octal) 16 (hexadecimal)
- Conversion of fractions
- Signed numbers, 2’s complement notation
- Basic arithmetic, addition, subtraction, overflow
  – (multiplication and division covered more later)
- BCD, ASCII, Parity







Computer word	Even parity	Odd parity
* 001010100	Set * to 1	Set * to 0
* 001110100	Set * to 0	Set * to 1

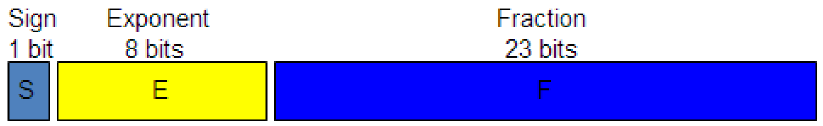


Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n

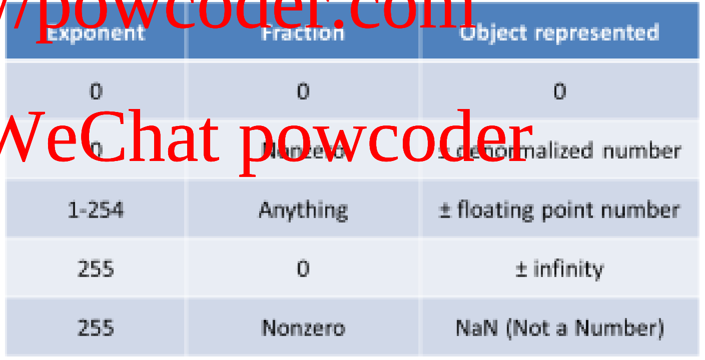
Number Representation

L3, PH 3.5

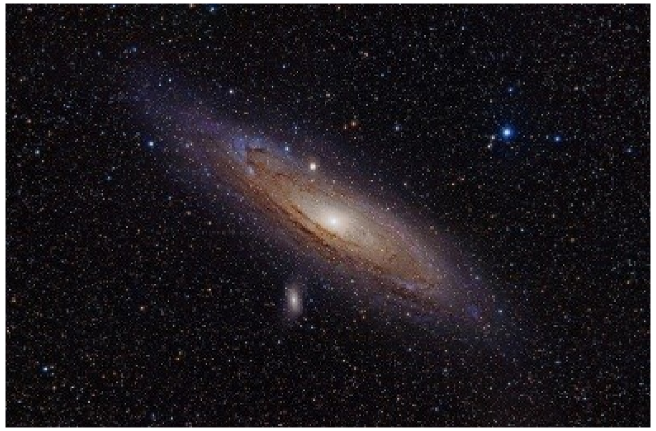
- IEEE Floating Point

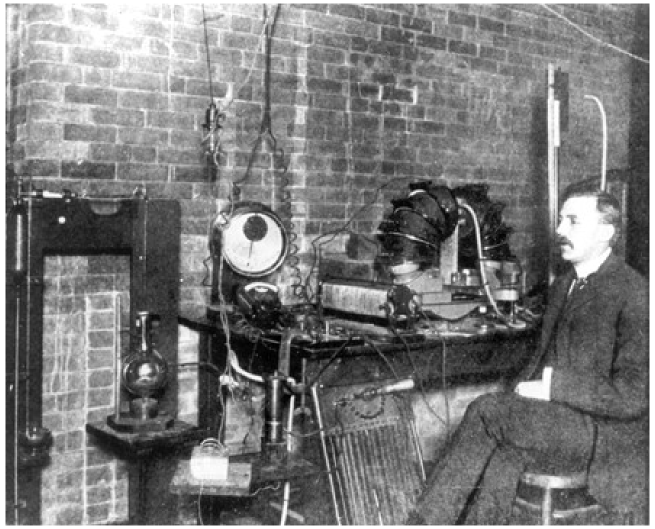


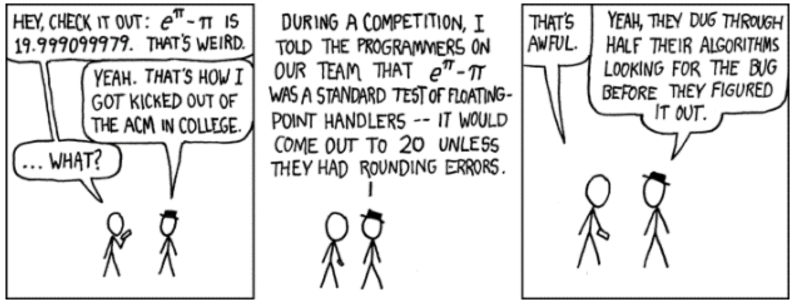
  - Normalization to get scientific notation
  - Sign, biased exponent, fractional part (mantissa)
  - Single, double, precision
  - Special numbers
    - +/- infinity, NaN
    - denormalized numbers
  - Addition, Multiplication
  - Rounding of floating point numbers (discussed again in a later class)



Exponent	Fraction	Object represented
0	0	0
0	Nonzero	± denormalized number
1-254	Anything	± floating point number
255	0	± infinity
255	Nonzero	NaN (Not a Number)



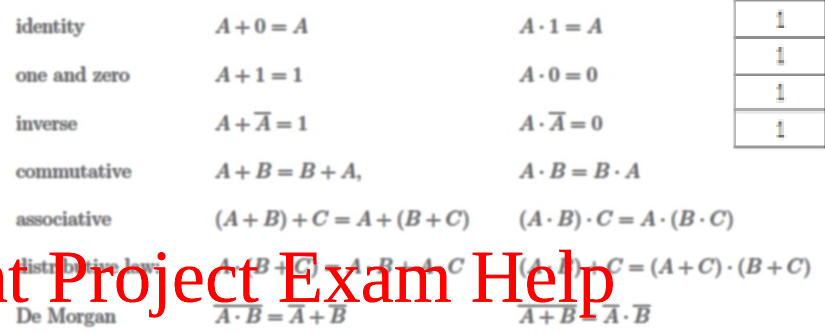



HEY, CHECK IT OUT: $e^{\pi}-\pi$ IS 19.999099979. THAT'S WEIRD.
YEAH. THAT'S HOW I GOT KICKED OUT OF THE ACM IN COLLEGE.
... WHAT?
DURING A COMPETITION, I TOLD THE PROGRAMMERS ON OUR TEAM THAT $e^{\pi}-\pi$ WAS A STANDARD TEST OF FLOATING-POINT HANDLERS -- IT WOULD COME OUT TO 20 UNLESS THEY HAD ROUNDING ERRORS.
THAT'S AWFUL.
YEAH, THEY DUG THROUGH HALF THEIR ALGORITHMS LOOKING FOR THE BUG BEFORE THEY FIGURED IT OUT.

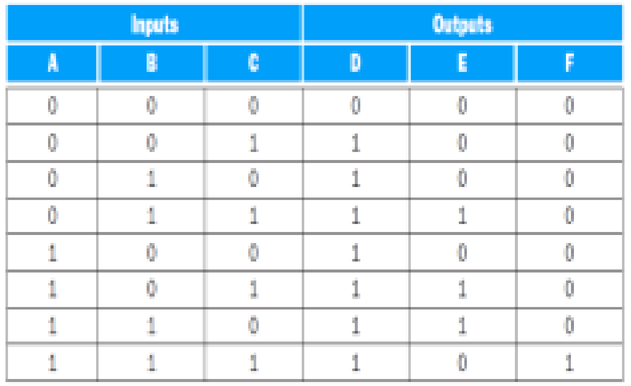
Boolean Algebra and Digital Circuits

L4, PH C.1-C.3

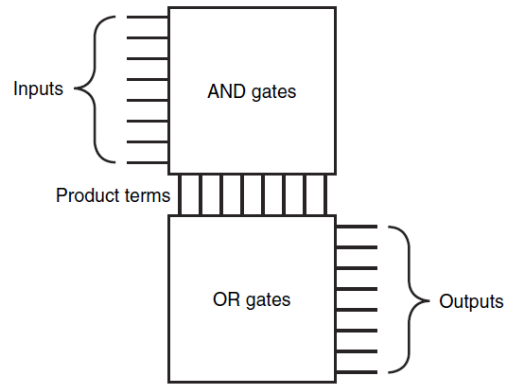
- Laws of Boolean algebra
- Algebraic simplification
- Truth tables / Don't Cares
- Sum of Products / Product of Sums / PLA
- Design of simple arithmetic circuits
  - Digital circuit gates: AND, OR, NOT, XOR
  - Half adder, Full adder, Subtraction
  - Encoders, Decoders, Multiplexors (L5)
- Circuit Minimization, hard ***(not on exam)***

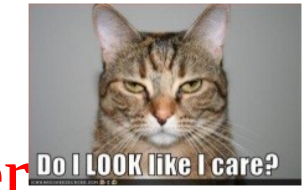


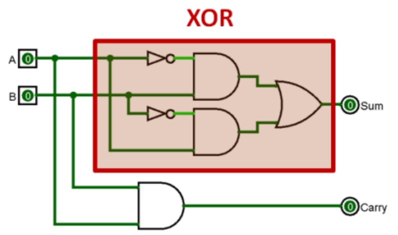
identity	$A+0=A$	$A\cdot 1=A$
one and zero	$A+1=1$	$A\cdot 0=0$
inverse	$A+\overline{A}=1$	$A\cdot\overline{A}=0$
commutative	$A+B=B+A,$	$A\cdot B=B\cdot A$
associative	$(A+B)+C=A+(B+C)$	$(A\cdot B)\cdot C=A\cdot(B\cdot C)$
distributive laws	$A\cdot(B+C)=A\cdot B+A\cdot C$	$(A\cdot B)+C=(A+C)\cdot(B+C)$
De Morgan	$\overline{A\cdot B}=\overline{A}+\overline{B}$	$\overline{A+B}=\overline{A}\cdot\overline{B}$

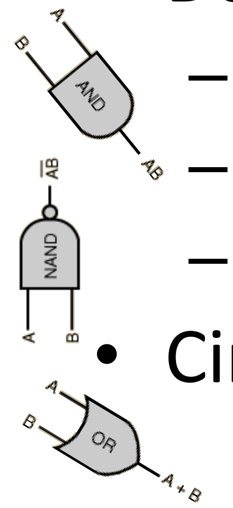


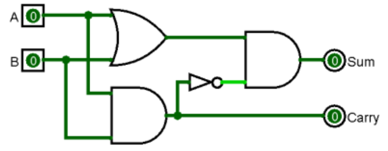
Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

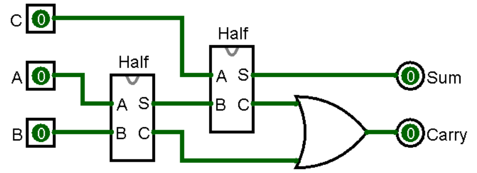








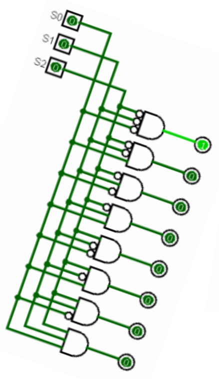
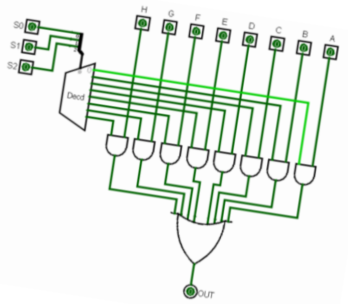
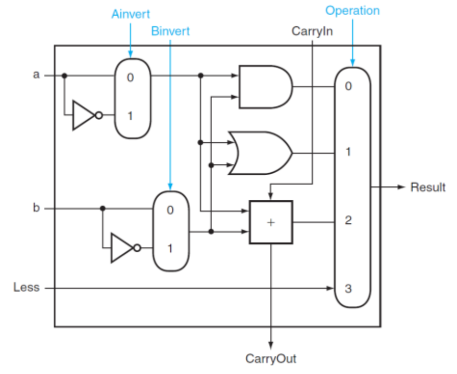


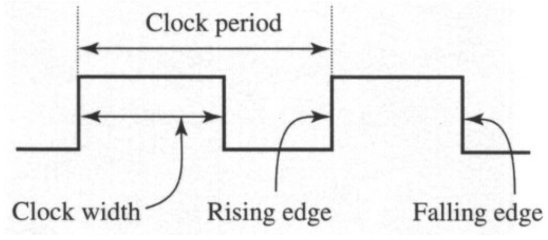


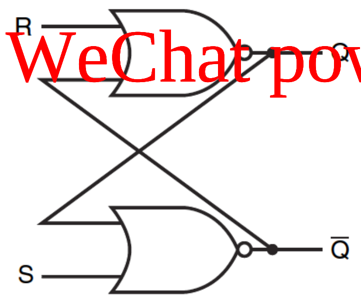
More, and Sequential Circuits

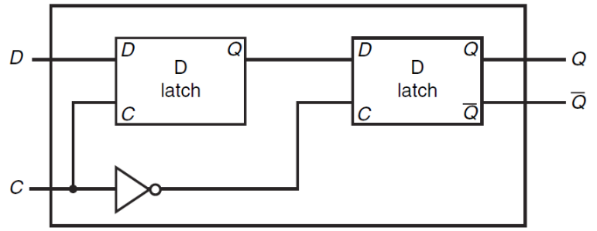
L5, C.7-C.8

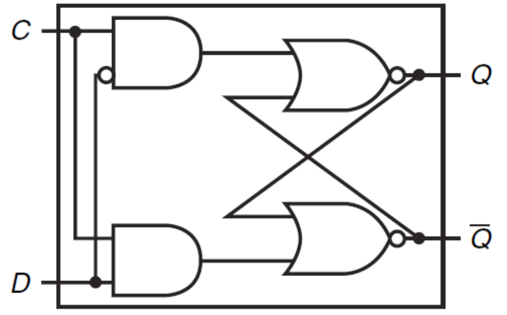
- Decoder, Encoder, MUX, ALU
- Combinational versus sequential circuits
- Clocks, Timing Diagrams (how inputs propagate to outputs)
- Sequential Circuits
  - SR latch,
  - D latch,
  - D flip-flop,
  - Toggle Flip Flop

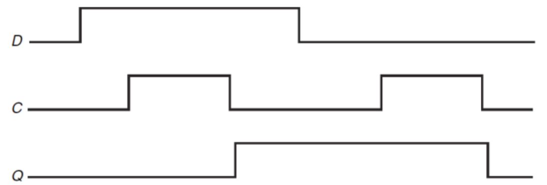




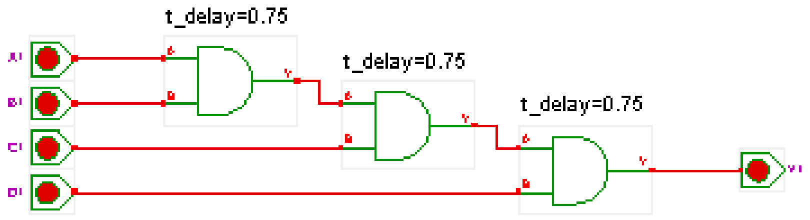








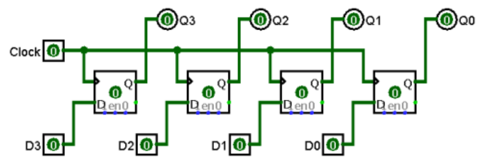


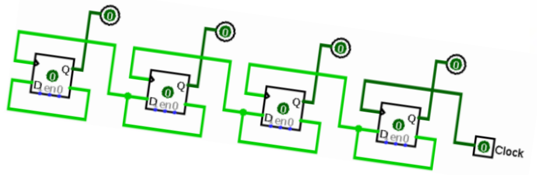


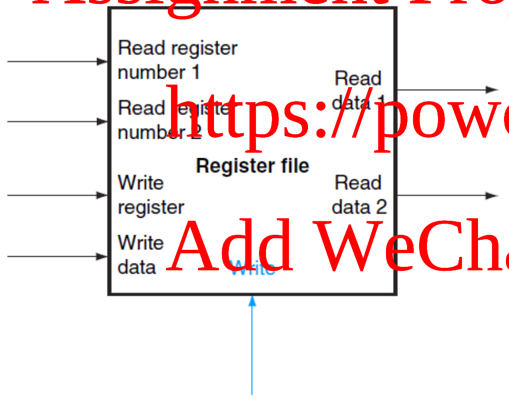
Registers and Memory

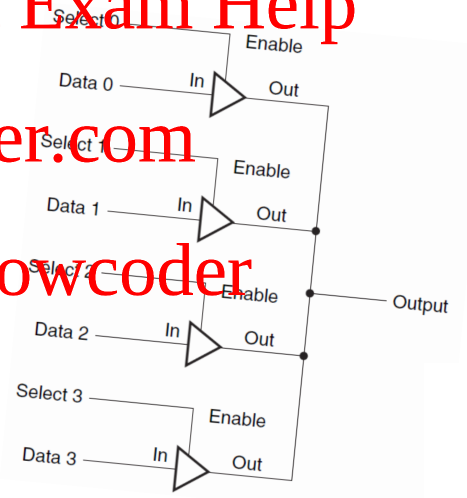
L6, C.9

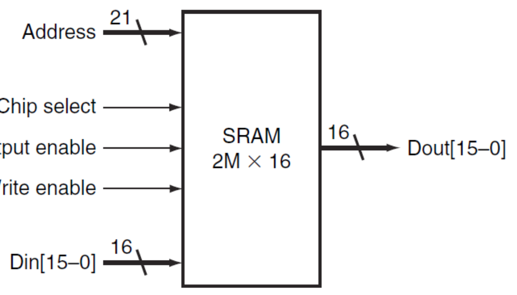
- Registers (shift registers, count registers)
  - Count up, count down, shift left, shift right
- Register File
- Memory
  - SRAM
  - DRAM
  - Tri-state buffers
  - Synchronous RAM, DDR RAM

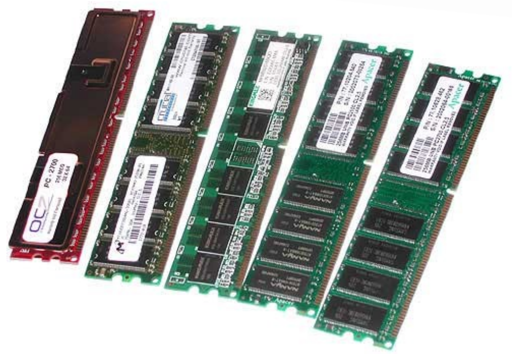


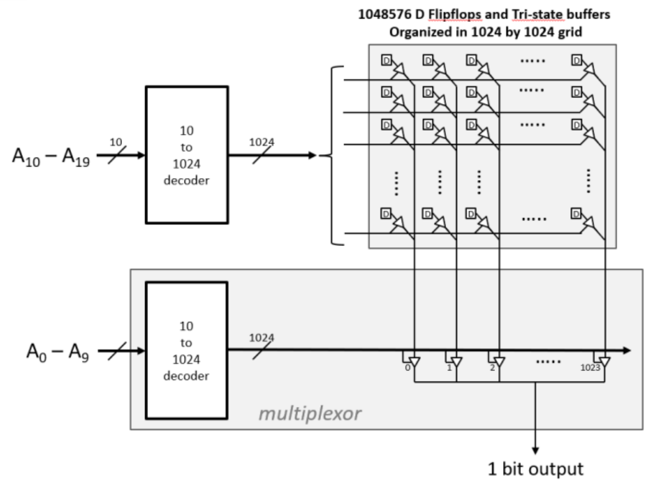










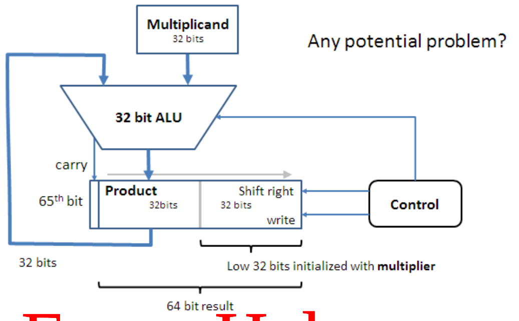


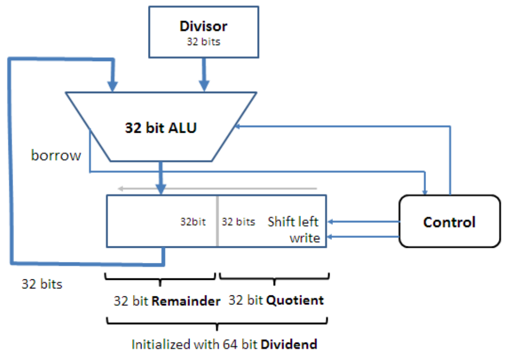
Multiplication and Division / FSMs

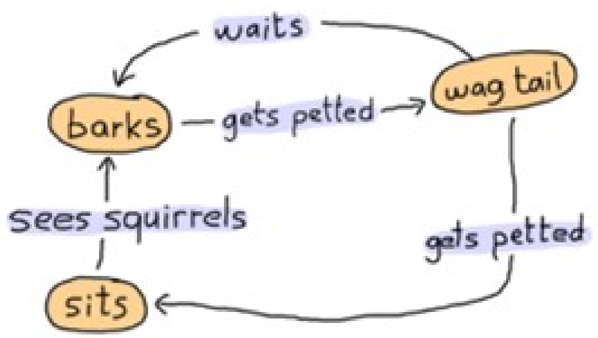
NOT ON EXAM

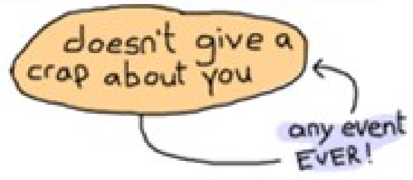
L7, PH 3.3-3.4 (again), C.10

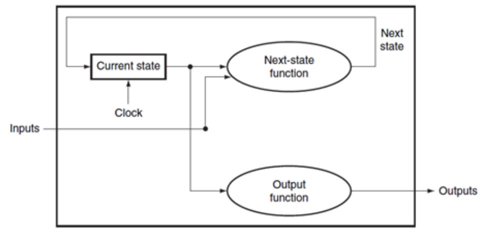
- Sequential multiplication circuit (two versions)
  - Shift registers, and control logic
- Sequential division circuit (two versions)
  - Issues with signed division
- Finite State Machines
  - *Moore* Machine vs Mealy Machine
  - Transition and output functions
  - Traffic light example, and others

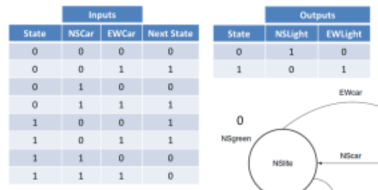






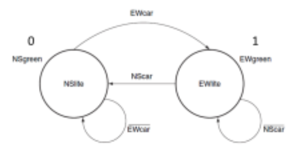






	Inputs		
State	NSCar	EWCar	Next State
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

	Outputs	
State	NSLight	EWLight
0	1	0
1	0	1

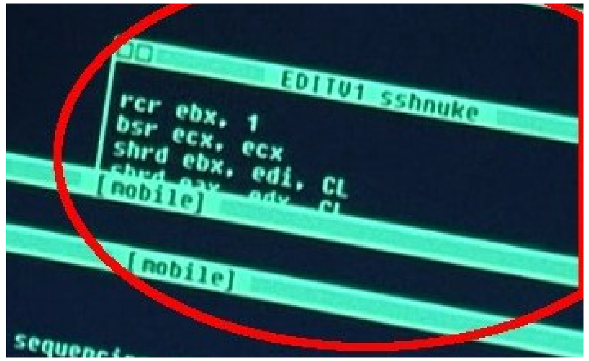


MIPS arithmetic and memory access

L8, PH 2.1-2.3

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

- Assembly operands are registers
- Addition and subtraction in MIPS
  – add, sub
- The 32 MIPS registers
- Instructions with an immediate
  – addi
- Data transfer from registers to memory and vice-versa.
  – The “lw” instruction and its syntax
  – The “sw” instruction and its syntax
- Byte-addressable memory
  – *Recall L13 wrt big/little endian!!*



```
rcr ebx, 1
bsr ecx, ecx
shrd ebx, edi, CL
```

MIPS assembly decisions

L9, PH 2.7

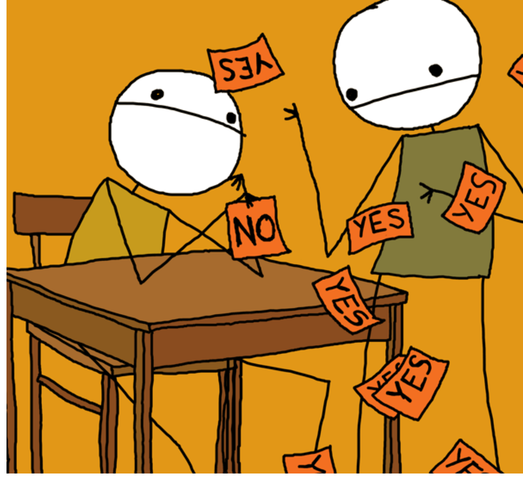
- Assembly operands are registers
- If-else statements using:
  - beq, bne, j <label>
- Loops using:
  - slt
- Inequalities using:
  - slt, beq, bne
- Case statements
- slt, slti, sltu, sltiu

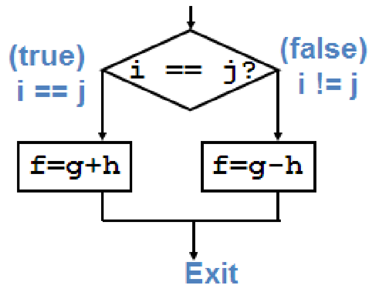
```
slt $t0,$s0,$s1 # $t0 = 1 if g < h
bne $t0,$0,Less # if (g < h) goto Less

slt $t0,$s1,$s0 # $t0 = 1 if g > h
bne $t0,$0,Grtr # if (g > h) goto Grtr

slt $t0,$s0,$s1 # $t0 = 1 if g < h
beq $t0,$0,Gteq # if (g >= h) goto Gteq

slt $t0,$s1,$s0 # $t0 = 1 if g > h
beq $t0,$0,Lteq # if (g <= h) goto Lteq
```





MIPS procedures

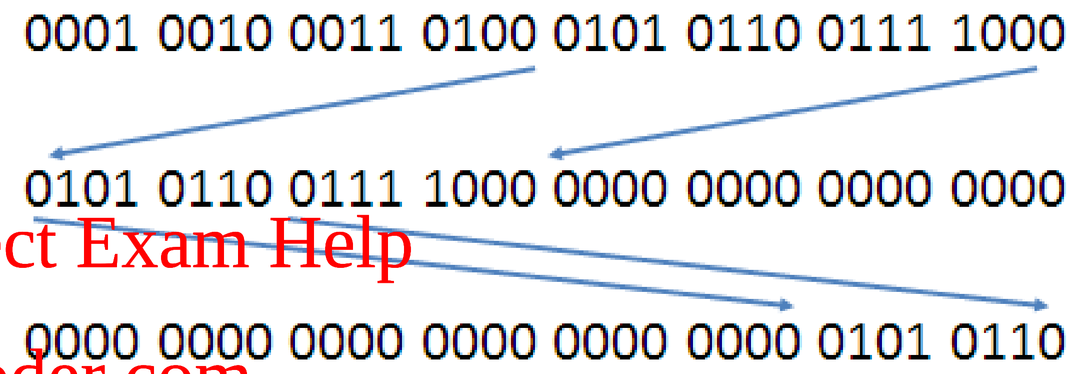
L10, PH 2.8, B.5, B.6

- Memory layout and the stack
- ***Register conventions***
  - Return address `$ra`
  - Arguments `$a0, $a1, $a2, $a3`
  - Return value `$v0, $v1`
  - Local variables `$s0, $s1, … , $s7`
- jal, jr
- Nested procedures
- MIPS naïve mult example
  - see `sort` example in PH 2.13 for more

Logical operations, shifts, arithmetic

L11, PH 2.6

- Logical
  - and, or, nor, andi, ori
- Shifts
  - sll, srl, sra
- Masking bits and setting bits
- Image colour component swapping example



MIPS Instruction Representation

L12-L13, PH 2.5, 2.10 (B.9, B.10)

R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		
J	opcode	target address				

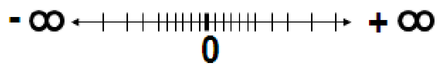
- R-format, I-format, J-format
  - rs, rt, rd, opcode, funct, shamt, immediate
- I-format limitation solved with lui
- PC-relative addressing, J-format addressing
- Disassembly
- Pseudo-instructions vs True instructions
  - move, li, ror, nop *(PH 3.9 discusses briefly)*
- Organization of an assembly program,
  - Data declarations, System calls
- ***Little Endian*** (LSB (least significant byte) in lowest) VS ***Big Endian*** (MSB in lowest)

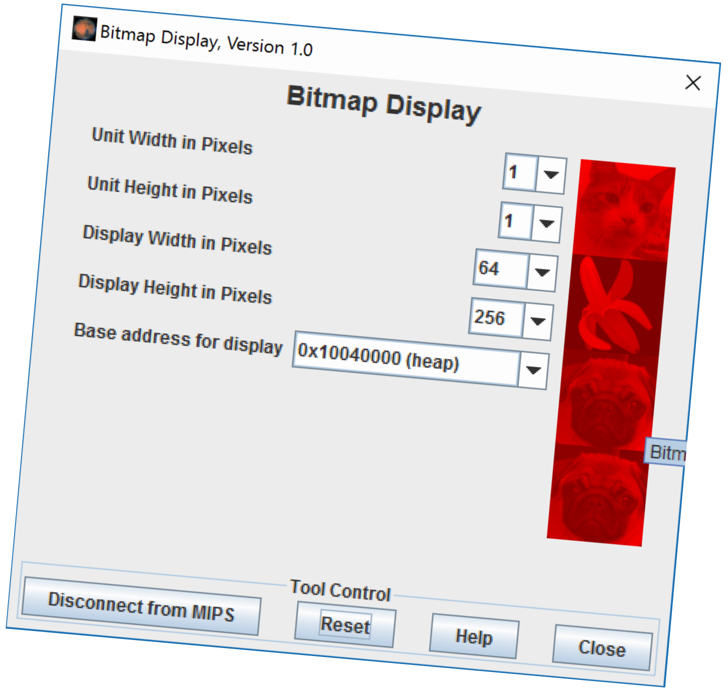
```
slti $t0,$a0,1
beq $t0,$zero,L1
addi $v0,$zero,1
addi $sp,$sp,8
jr $ra
```

MIPS Integer Mult/Div & Floating Point

L14, PH 3.3-3.5

- Integer Multiplication and Division
  - mul, mult, div, divu, mfhi, mflo
- Floating point addition subtraction multiplication
  - add.s, sub.s, mul.s, add.d, sub.d, mul.d, div.d
- Coprocessor commands, mfc1, mtc1
- Closer look at denormalized numbers
- Closer look at rounding modes
  - (up, down, truncate, even)
- Non-Associativity of floating point
- Fahrenheit to Celsius example, and A3




Bitmap Display, Version 1.0

Bitmap Display

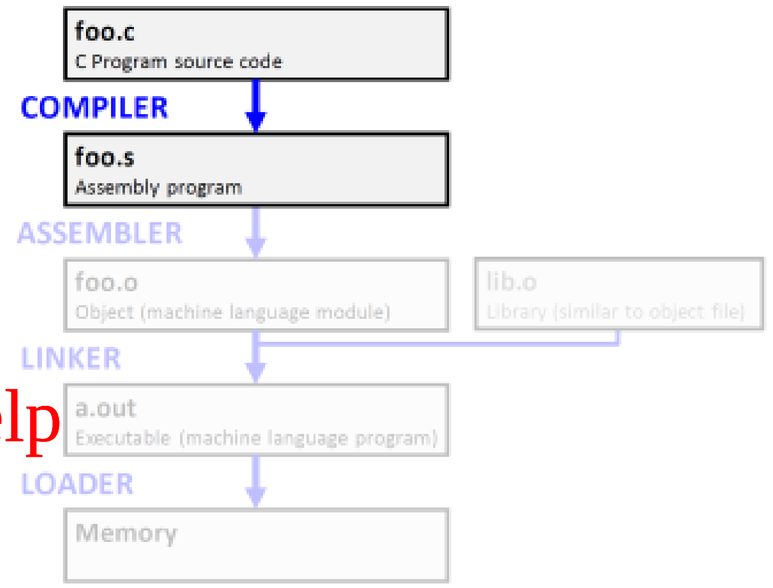
Unit Width in Pixels: 1
Unit Height in Pixels: 1
Display Width in Pixels: 64
Display Height in Pixels: 256
Base address for display: 0x10040000 (heap)

Assembling, Linking and Loading

L15, PH 2.12 B.1-B.4

- Assembler
  - Directives
  - Pseudo-instruction replacement
  - Creates object file
  - Symbol Table, Relocation Table
- Linker (Link-Editor)
  - Combines object files into a module
  - Resolves references
- Loader
- A detailed example

Steps to starting a program


foo.c
C Program source code
COMPILER
foo.s
Assembly program
ASSEMBLER
foo.o
Object (machine language module)
lib.o
Library (similar to object file)
LINKER
a.out
Executable (machine language program)
LOADER
Memory

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

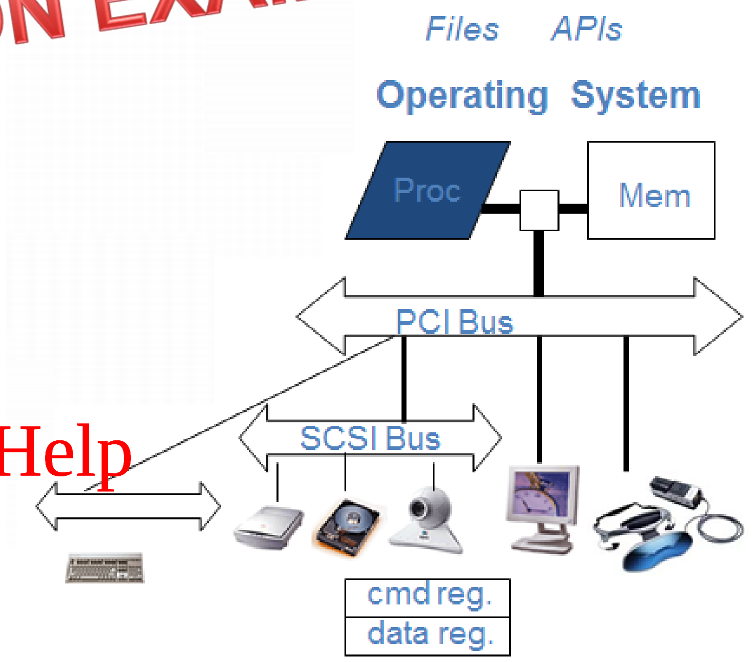
```
0x000000  00100111101111011111111111100000
0x000004  10101111101111110000000000010100
0x000008  10101111101001000000000000100000
0x00000c  10101111101001010000000000100100
0x000010  10101111101000000000000000011000
0x000014  10101111101000000000000000011100
0x000018  10001111101011100000000000011100
0x00001c  00000001110011100000000000011001
0x000020  00000000000000000111100000010010
0x000024  10001111101110000000000000011000
0x000028  00000011000011111100100000100001
0x00002c  10101111101010000000000000011100
0x000030  00100101110010000000000000000001
0x000034  10101111101110010000000000011000
0x000038  00101001000000010000000001100101
0x00003c  00010100001000001111111111110111
0x000040  00111100000001000001000000000000
0x000044  00110100100001000000010000110000
0x000048  10001111101001010000000000011000
0x00004c  00001100000100000000000011101100
0x000050  00000000000000000000100000100001
0x000054  10001111101111110000000000010100
0x000058  00100111101111010000000000100000
0x00005c  00000011111000000000000000001000
```

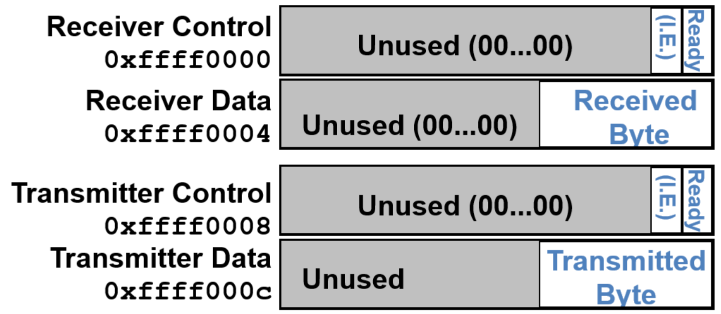
I/O Polling and Interrupts

L16, *(PH 6.5-6.7, B.7,B.8)*

NOT ON EXAM

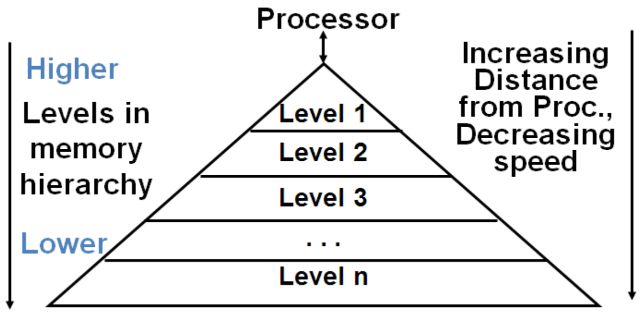
- I/O background, speed mismatches between processor, memory, devices
- Memory-mapped I/O and polling costs
- MARS I/O simulation, Receiver, Transmitter
  - Control (command) register, Data register
  - Control strategy (ready bit, received/transmitted byte)
  - I/O example (keyboard and terminal)
- Interrupt I/O: save PC, jump to service routine, perform transfer, then return
- Portion of MIPS architecture for interrupts called "coprocessor 0", instructions and registers:
  - Data transfer(lwc0, swc0) Move (mfc0, mtc0)
  - Status $12 Interrupt enable, Cause $13 Exception type, EPC $14 Return address





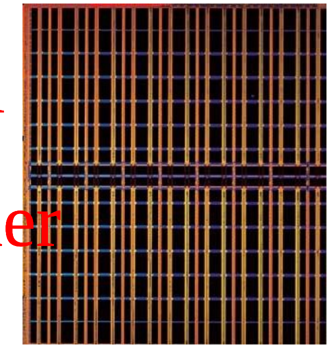
The Memory Hierarchy – Caches Part 1

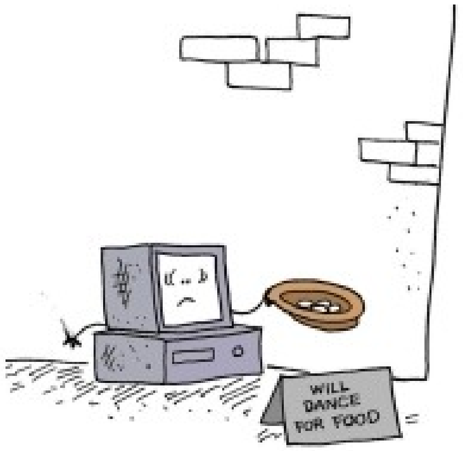
L18, PH 5.1-5.3

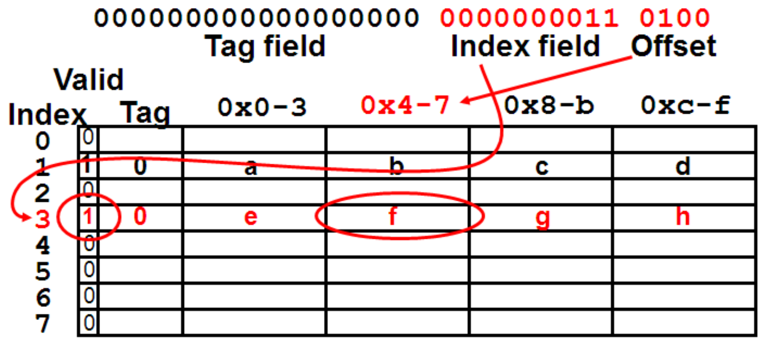


Processor
Higher
Levels in memory hierarchy
Lower
Level 1
Level 2
Level 3
. . .
Level n
Increasing Distance from Proc., Decreasing speed

- Levels of the memory hierarchy
- General behaviour as you go from level 1 to level n (increased distance from processor)
- Caches
  - Notion of cache size versus block size
  - Direct-mapped cache: tag, index, offset
  - Detailed example of accessing data in a direct-mapped cache
  - Big picture: spatial and temporal locality



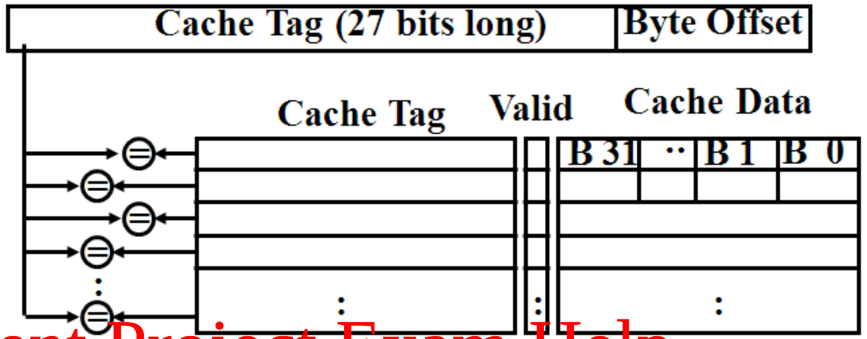


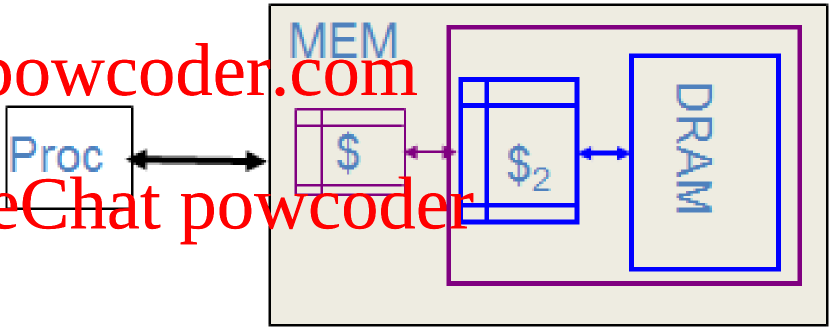


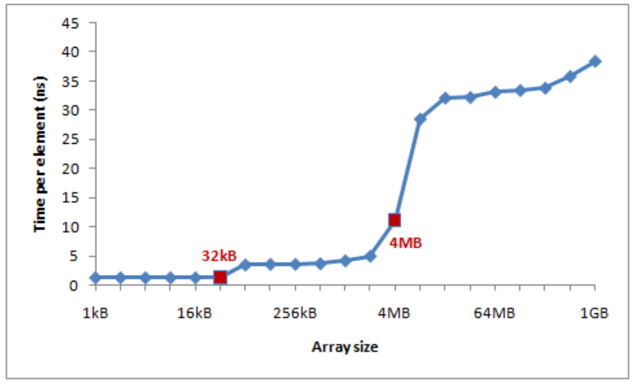
The Memory Hierarchy – Caches Part 2

L19, PH 5.1-5.3

- Block Size Tradeoff
- Types of Cache Misses
- Average Access Time
- Fully Associative Cache
- N-Way Associative Cache
  - A detailed example
- Block Replacement Policy
  - Random Versus LRU
- Multilevel Caches
- Cache write policy
  - Write-thru versus write-back
- Cache in action java demos



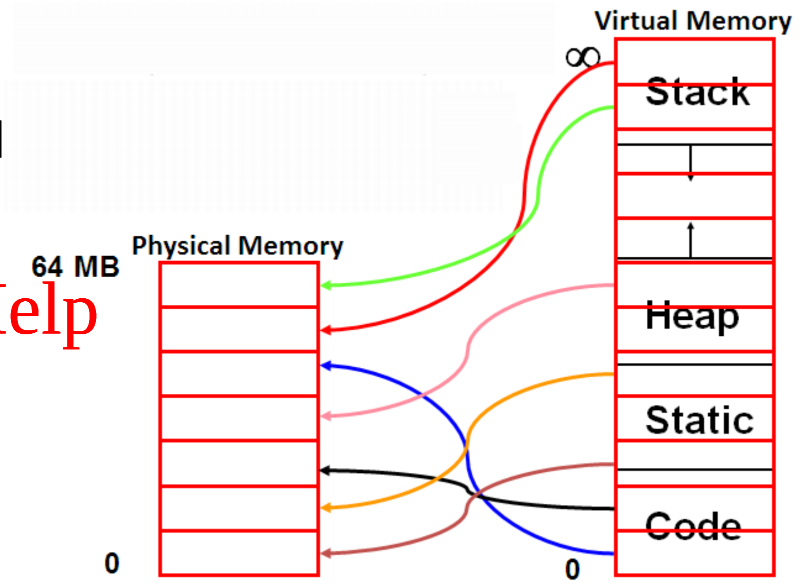


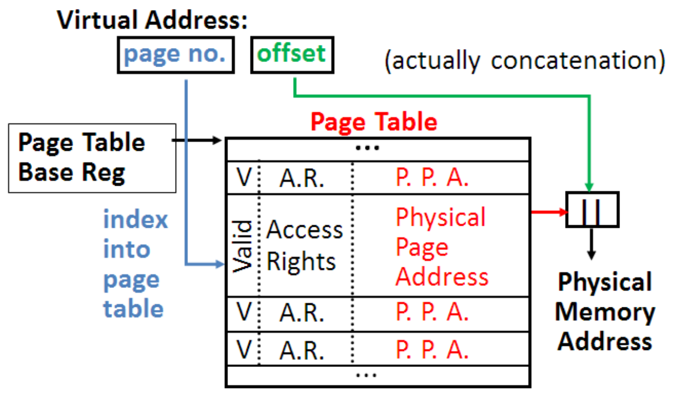


Virtual Memory Part 1

L20, PH 5.4, 5.5

- Mapping Physical Memory to Virtual Memo
  - Page no. and offset
  - Page Table
  - Calculation of physical address
  - Page Table Base register
- Virtual Memory Problems
  - Indirection to calculate physical address is slow
    - Use a TLB (a cache of recently used translations)
  - Not enough RAM
  - Too much space used to store page tables
    - Multi-level page tables ***(not on exam)***

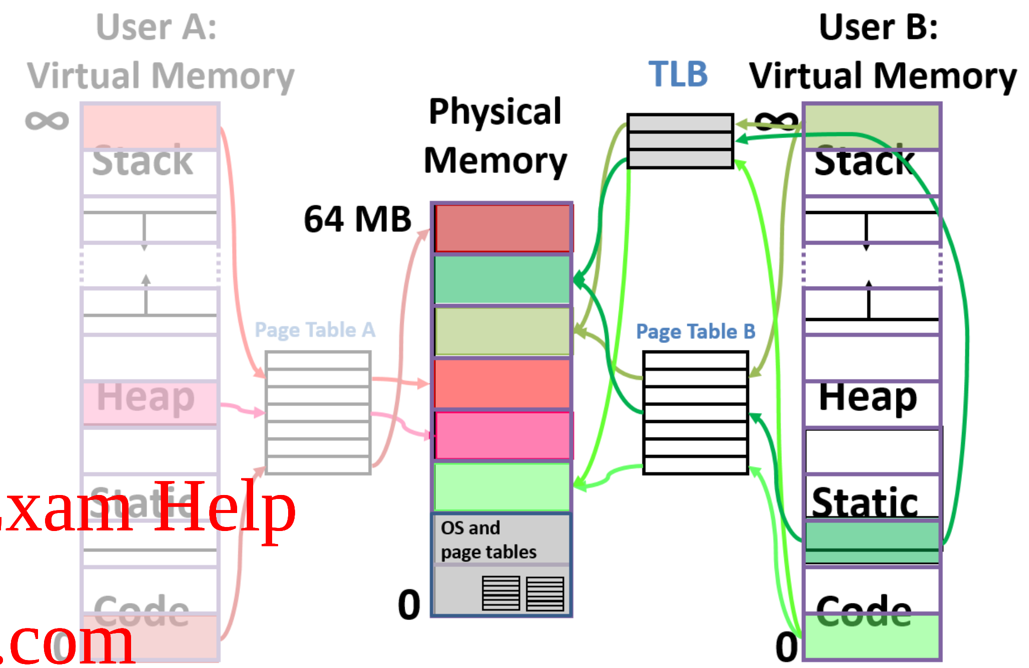


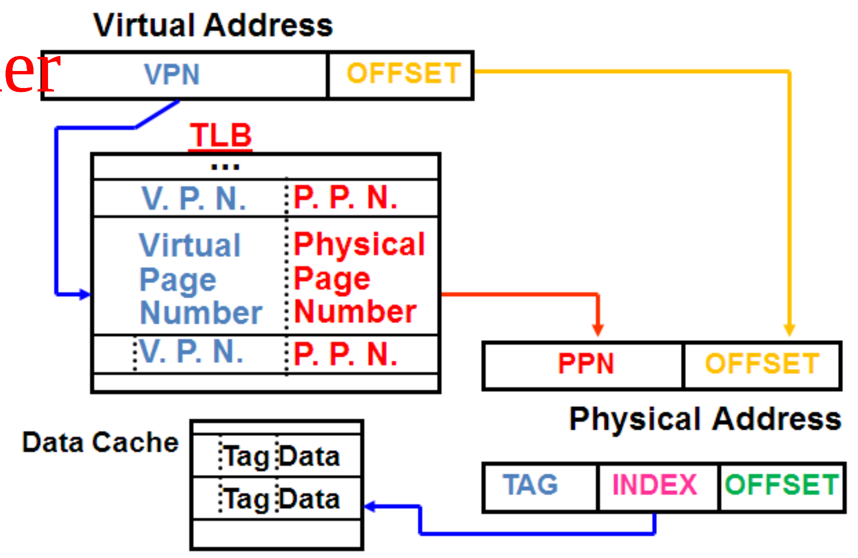


Virtual Memory Part 2

L21, PH 5.4, 5.5

- The advantages provided by virtual memory
  - translation, protection, sharing
- The overall process:
  - Check TLB (input: VPN, output: PPN)
    - hit: fetch translation
    - miss: check page table (in memory)
      - **Page table hit: fetch translation**
      - **Page table miss: page fault, fetch page from disk to memory, return translation to TLB**
  - Check cache (input: PPN, output: data)
    - hit: return value
    - miss: fetch value from memory

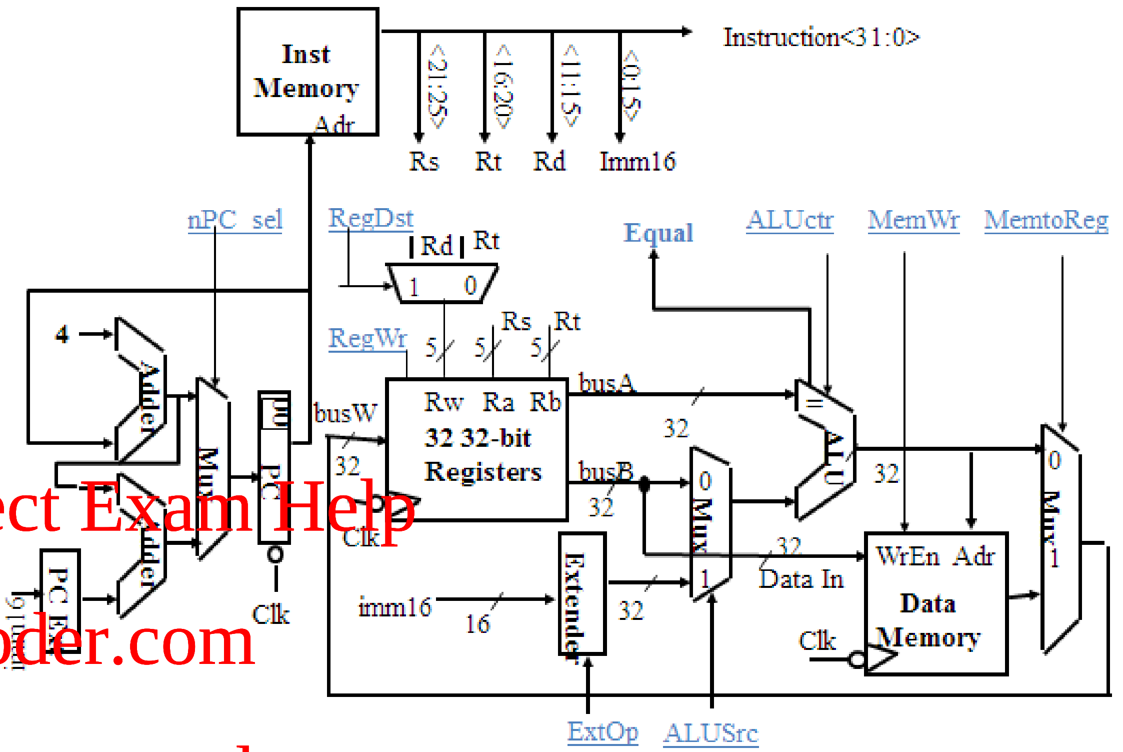




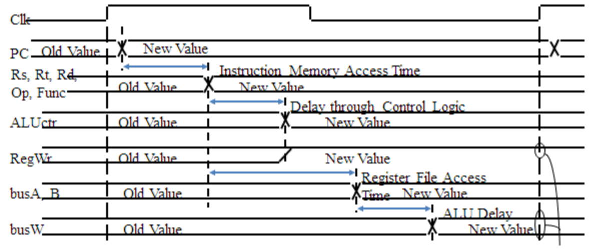
Single Cycle CPU Datapath

L22, PH 4.1-4.3

- Design of a Processor
  - Instruction set architecture
  - Datapath Requirements
  - Establish clocking
  - Assemble datapath
  - Determine control settings
  - Assemble control logic
- Register transfer language
- Example with reduced MIPS instruction set
- Register-Register timing for one complete cycle



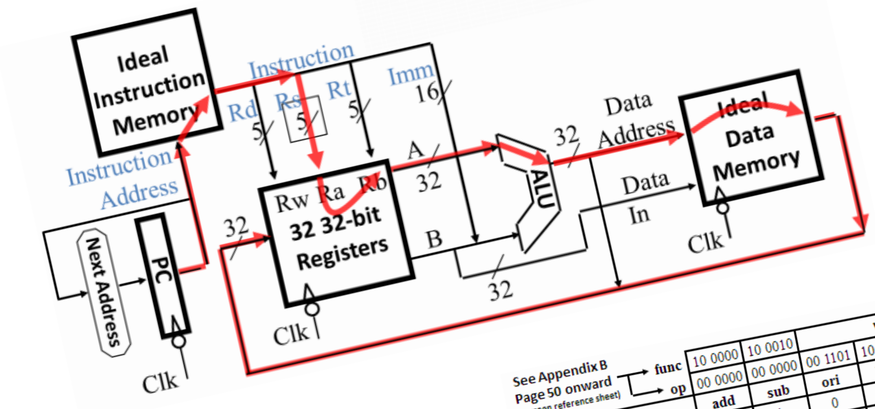
```
ADDU   R[rd] = R[rs] + R[rt]; ...
SUBU   R[rd] = R[rs] – R[rt]; ...
ORI    R[rt] = R[rs] | zero_ext(Imm16)...
BEQ    if ( R[rs] == R[rt] )...
```

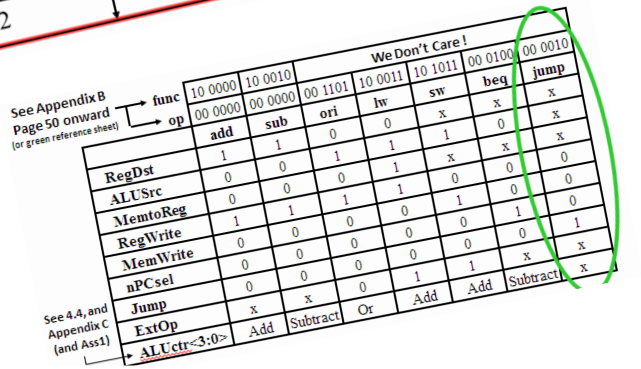


Single Cycle CPU Control

L23, PH 4.4

- **Critical Path** of single cycle CPU
- Meaning of Control Signals
- Instruction Fetch Unit
- **Control Signals** for MIPS-Lite
  - – Truth table with don't care
- Datapath during Branches and Jumps



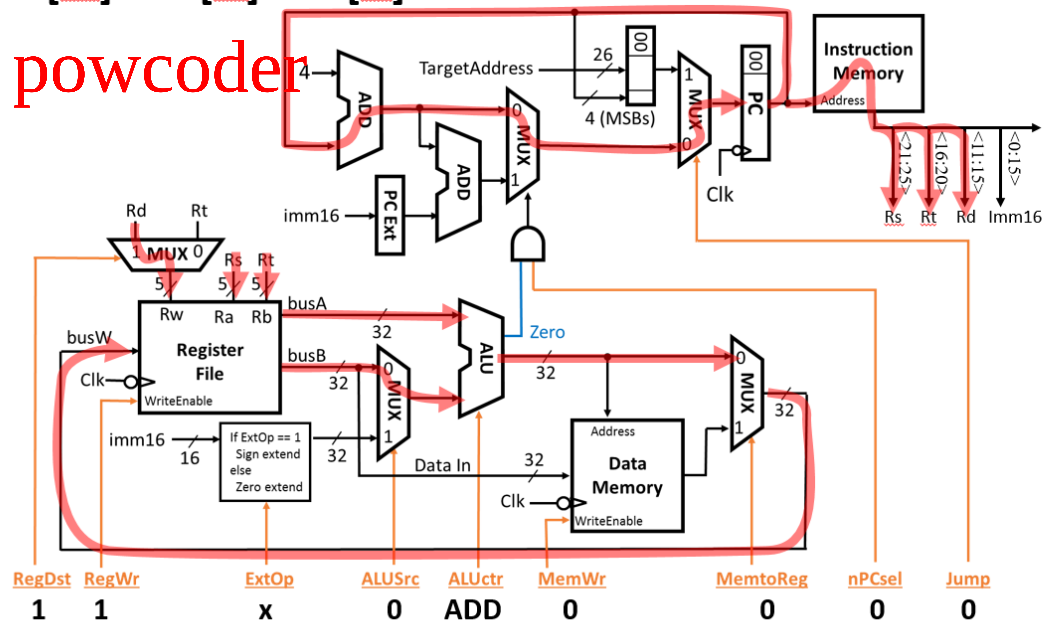


See Appendix B Page 50 onward (or green reference sheet)

See 4.4, and Appendix C (and Ass1)

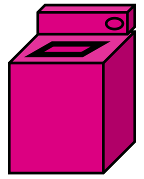
	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	We Don't Care !				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<3:0>	Add	Subtract	Or	Add	Add	Subtract	x

R[rd] = R[rs] + R[rt]



Pipelining

L24-L25, PH 4.5, 4.7, 4.8

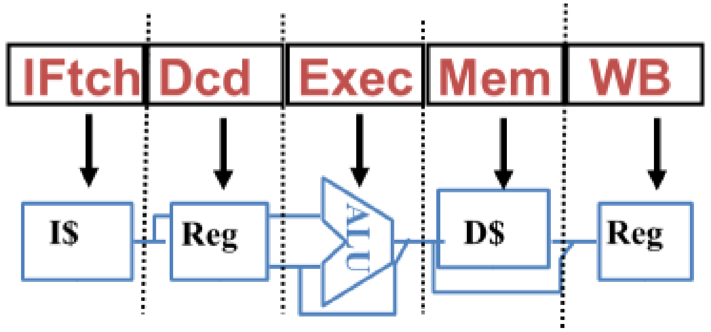


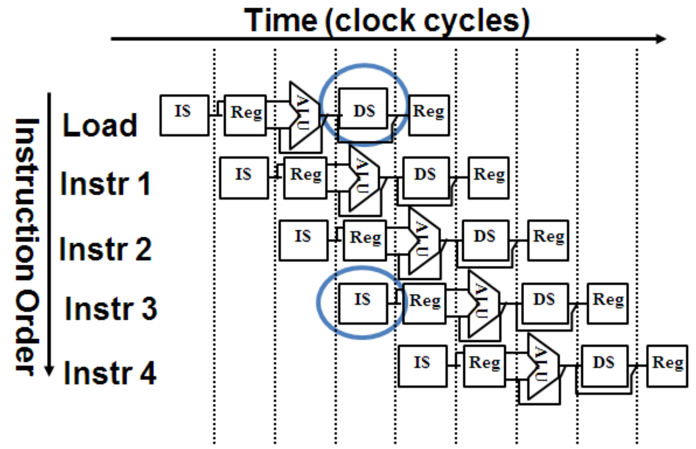
- The 5 stages of the datapath (laundry analogy)
  - Fetch, decode, execute, memory, writeback

- **Latency** vs **Throughput**

- Pipeline Hazards / Bubbles

  - **Structure** (shared resources, e.g., memory)
  - **Control** (e.g., branches)
  - **Data** (e.g., need results of previous instruction)
- Pipeline stalls / bubbles



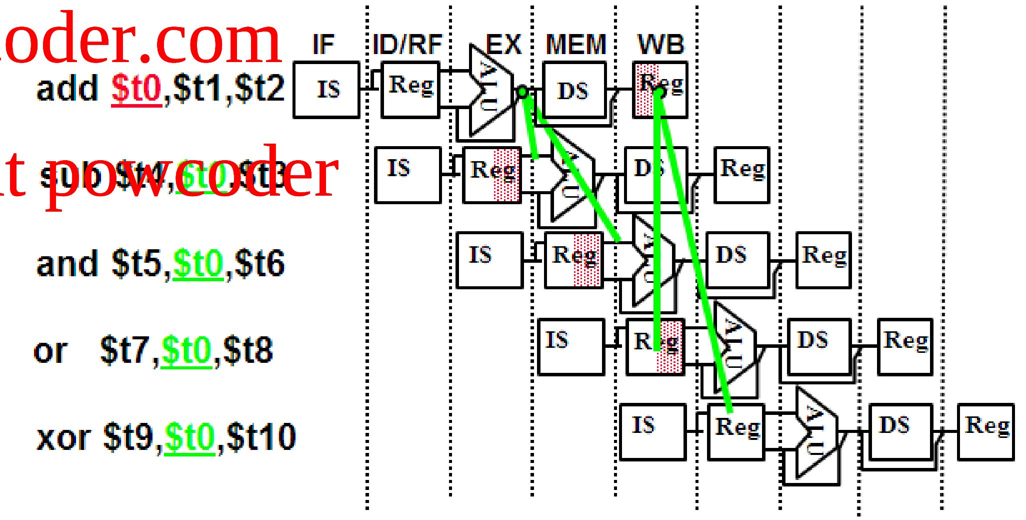


Pipelining

L24-L25, PH 4.5, 4.7, 4.8

```
loop: lw $t0, 0($s1)
      addiu $s1, $s1, -4
      addu $t0, $t0, $s2
      bne $s1, $0, loop
      sw $t0, 4($s1)
```

- Optimizations and addressing hazards
  - Data forwarding / bypass
  - Branch delay slot
  - Interlock
  - Load delay slot
  - Instruction reordering
  - Loop unrolling



Strategy for Reviewing Material

- Review all slides and review your notes
- Review the examples we did in lectures
- Review the relevant portions of the textbook (especially for anything you find unclear!)
- Post your questions to MyCourses general discussion

Sections in 4th edition

- Review the noted sections marked at the top of the slides in Patterson and Hennessey Computer Organization and Design 4th edition

```
1.1 1.2 1.3
2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8     2.10     2.12
3.1 3.2         3.5
4.1 4.2 4.3 4.4 4.5     4.7 4.8
5.1 5.2 5.3 5.4 5.5
B.1 B.2 B.3 B.4 B.5 B.6         B.9 (SPIM, not MARS) B.10
C.1 C.2 C.3         B.6 C.7 C.8 C.9
```

Book material will typically provide a less terse explanation than the slides / lectures, and may in some cases go into more depth.

Sections in 5th edition

- Review the noted sections marked at the top of the slides in Patterson and Hennessey Computer Organization and Design ***5th edition***

```
1.1 1.3 1.4     (1.2 also a nice review of big ideas)
2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8     2.10     2.12
3.1 3.2             3.5
4.1 4.2 4.3 4.4 4.5     4.7 4.8
5.1 5.3 5.4 5.5 5.6
A.1 A.2 A.3 A.4 A.5 A.6             A.9 (SPIM, not MARS) A.10
B.1 B.2 B.3             B.6 B.7 B.8 B.9
```

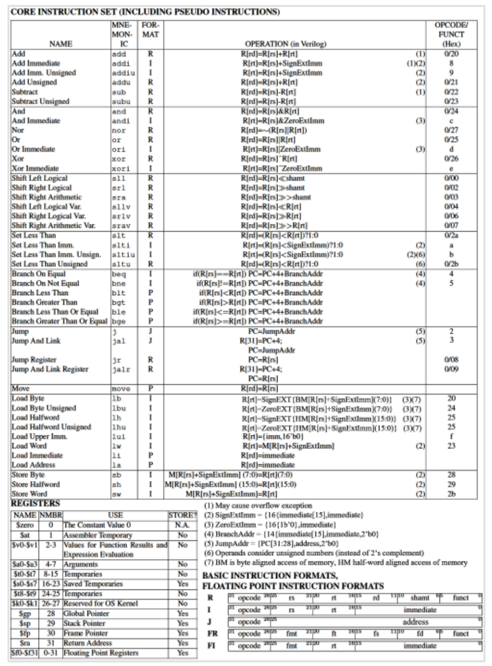
Book material will typically provide a less terse explanation than the slides / lectures, and may in some cases go into more depth.

(red highlights where section numbers differ in 5th edition)

(class schedule shows 6th edition sections which are also different)

A few other comments

- MIPS reference sheet, know how to use it!
- Exam will have multiple choice questions
  - Some questions are very easy and you will ***answer in seconds***
  - Other questions ***will be harder***, but **DON'T PANIC**! You have ***4.5 minutes on average*** per question!
- Unanswered questions are worth zero
  - If you do not know the answer, then make an educated guess among the 4 responses.



CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/FUNCT (Hex)
Add	`add`	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	`addi`	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	`addiu`	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	`addu`	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	`sub`	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	`subu`	R	R[rd]=R[rs]-R[rt]		0/23
And	`and`	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	`andi`	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	`nor`	R	R[rd]=~(R[rs]|R[rt])		0/27
Or	`or`	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	`ori`	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	`xor`	R	R[rd]=R[rs]^R[rt]		0/26
Xor Immediate	`xori`	I	R[rt]=R[rs]^ZeroExtImm		e
Shift Left Logical	`sll`	R	R[rd]=R[rs]<<shamt		0/00
Shift Right Logical	`srl`	R	R[rd]=R[rs]>>shamt		0/02
Shift Right Arithmetic	`sra`	R	R[rd]=R[rs]>>>shamt		0/03
Shift Left Logical Var.	`sllv`	R	R[rd]=R[rs]<<R[rt]		0/04
Shift Right Logical Var.	`srlv`	R	R[rd]=R[rs]>>R[rt]		0/06
Shift Right Arithmetic Var.	`srav`	R	R[rd]=R[rs]>>>R[rt]		0/07
Set Less Than	`slt`	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	`slti`	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	`sltiu`	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	`sltu`	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	`beq`	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	`bne`	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	`blt`	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	`bgt`	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	`ble`	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	`bge`	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	`j`	J	PC=JumpAddr	(5)	2
Jump And Link	`jal`	J	R[31]=PC+4; PC=JumpAddr	(5)	3
Jump Register	`jr`	R	PC=R[rs]		0/08
Jump And Link Register	`jalr`	R	R[31]=PC+4; PC=R[rs]		0/09
Move	`move`	P	R[rd]=R[rs]		
Load Byte	`lb`	I	R[rt]=SignEXT{BM[R[rs]+SignExtImm](7:0)}	(3)(7)	20
Load Byte Unsigned	`lbu`	I	R[rt]=ZeroEXT{BM[R[rs]+SignExtImm](7:0)}	(3)(7)	24
Load Halfword	`lh`	I	R[rt]=SignEXT{HM[R[rs]+SignExtImm](15:0)}	(3)(7)	25
Load Halfword Unsigned	`lhu`	I	R[rt]=ZeroEXT{HM[R[rs]+SignExtImm](15:0)}	(3)(7)	25
Load Upper Imm.	`lui`	I	R[rt]={imm,16'b0}		f
Load Word	`lw`	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	`li`	P	R[rd]=immediate		
Load Address	`la`	P	R[rd]=immediate		
Store Byte	`sb`	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	`sh`	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	`sw`	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

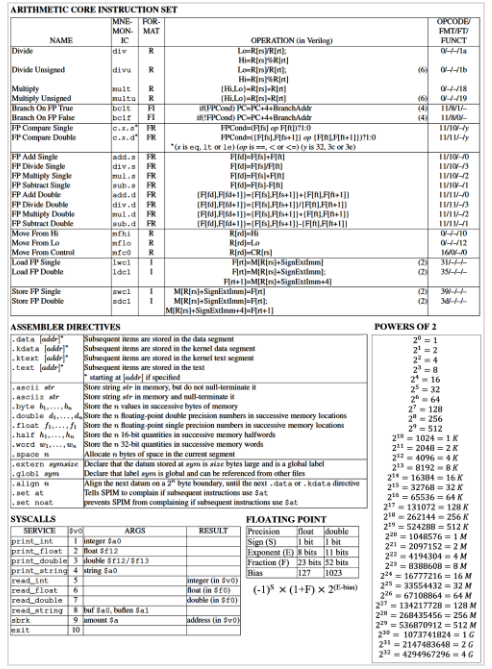
REGISTERS

NAME	NMBR	USE	STORE?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes
$f0-$f31	0-31	Floating Point Registers	Yes

(1) May cause overflow exception
(2) SignExtImm = {16{immediate[15],immediate}
(3) ZeroExtImm = {16{1b'0},immediate}
(4) BranchAddr = {14{immediate[15],immediate,2'b0}
(5) JumpAddr = {PC[31:28],address,2'b0}
(6) Operands consider unsigned numbers (instead of 2's complement)
(7) BM is byte aligned access of memory, HM half-word aligned access of memory

BASIC INSTRUCTION FORMATS, FLOATING POINT INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		
J	opcode	address				
FR	opcode	fmt	ft	fs	fd	funct
FI	opcode	fmt	rt	immediate		



ARITHMETIC CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/FMT/FT/FUNCT
Divide	`div`	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/–/–/1a
Divide Unsigned	`divu`	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/–/–/1b
Multiply	`mult`	R	{Hi,Lo}=R[rs]*R[rt]		0/–/–/18
Multiply Unsigned	`multu`	R	{Hi,Lo}=R[rs]*R[rt]	(6)	0/–/–/19
Branch On FP True	`bc1t`	FI	if(FPCond) PC=PC+4+BranchAddr	(4)	11/8/1/–
Branch On FP False	`bc1f`	FI	if(!FPCond) PC=PC+4+BranchAddr	(4)	11/8/0/–
FP Compare Single	`c.x.s`*	FR	FPCond=(F[fs] *op* F[ft])?1:0		11/10/–/y
FP Compare Double	`c.x.d`*	FR	FPCond=({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]})?1:0		11/11/–/y
			*(*x* is `eq`, `lt` or `le`) (*op* is ==, < or <=) (*y* is 32, 3c or 3e)		
FP Add Single	`add.s`	FR	F[fd]=F[fs]+F[ft]		11/10/–/0
FP Divide Single	`div.s`	FR	F[fd]=F[fs]/F[ft]		11/10/–/3
FP Multiply Single	`mul.s`	FR	F[fd]=F[fs]*F[ft]		11/10/–/2
FP Subtract Single	`sub.s`	FR	F[fd]=F[fs]-F[ft]		11/10/–/1
FP Add Double	`add.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}+{F[ft],F[ft+1]}		11/11/–/0
FP Divide Double	`div.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}/{F[ft],F[ft+1]}		11/11/–/3
FP Multiply Double	`mul.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}*{F[ft],F[ft+1]}		11/11/–/2
FP Subtract Double	`sub.d`	FR	{F[fd],F[fd+1]}={F[fs],F[fs+1]}-{F[ft],F[ft+1]}		11/11/–/1
Move From Hi	`mfhi`	R	R[rd]=Hi		0/–/–/10
Move From Lo	`mflo`	R	R[rd]=Lo		0/–/–/12
Move From Control	`mfc0`	R	R[rd]=CR[rs]		16/0/–/0
Load FP Single	`lwc1`	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/–/–/–
Load FP Double	`ldc1`	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/–/–/–
Store FP Single	`swc1`	I	M[R[rs]+SignExtImm]=F[rt]	(2)	39/–/–/–
Store FP Double	`sdc1`	I	M[R[rs]+SignExtImm]=F[rt]; M[R[rs]+SignExtImm+4]=F[rt+1]	(2)	3d/–/–/–

ASSEMBLER DIRECTIVES

`.data` [*addr*]*	Subsequent items are stored in the data segment
`.kdata` [*addr*]*	Subsequent items are stored in the kernel data segment
`.ktext` [*addr*]*	Subsequent items are stored in the kernel text segment
`.text` [*addr*]*	Subsequent items are stored in the text
	* starting at [*addr*] if specified
`.ascii` *str*	Store string *str* in memory, but do not null-terminate it
`.asciiz` *str*	Store string *str* in memory and null-terminate it
`.byte` $b_1, \ldots, b_n$	Store the *n* values in successive bytes of memory
`.double` $d_1, \ldots, d_n$	Store the *n* floating-point double precision numbers in successive memory locations
`.float` $f_1, \ldots, f_1$	Store the *n* floating-point single precision numbers in successive memory locations
`.half` $h_1, \ldots, h_n$	Store the *n* 16-bit quantities in successive memory halfwords
`.word` $w_1, \ldots, w_n$	Store the *n* 32-bit quantities in successive memory words
`.space` *n*	Allocate *n* bytes of space in the current segment
`.extern` *symsize*	Declare that the datum stored at *sym* is *size* bytes large and is a global label
`.globl` *sym*	Declare that label *sym* is global and can be referenced from other files
`.align` *n*	Align the next datum on a 2^n byte boundary, until the next `.data` or `.kdata` directive
`.set at`	Tells SPIM to complain if subsequent instructions use `$at`
`.set noat`	prevents SPIM from complaining if subsequent instructions use `$at`

SYSCALLS

SERVICE	`$v0`	ARGS	RESULT
`print_int`	1	integer `$a0`	
`print_float`	2	float `$f12`	
`print_double`	3	double `$f12/$f13`	
`print_string`	4	string `$a0`	
`read_int`	5		integer (in `$v0`)
`read_float`	6		float (in `$f0`)
`read_double`	7		double (in `$f0`)
`read_string`	8	buf `$a0`, buflen `$a1`	
`sbrk`	9	amount `$a`	address (in `$v0`)
`exit`	10		

FLOATING POINT

Precision	float	double
Sign (S)	1 bit	1 bit
Exponent (E)	8 bits	11 bits
Fraction (F)	23 bits	52 bits
Bias	127	1023

$$(-1)^S \times (1+F) \times 2^{(E\text{-bias})}$$

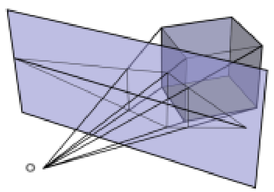
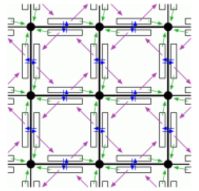
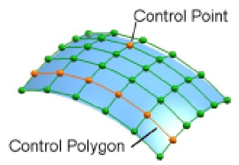
POWERS OF 2

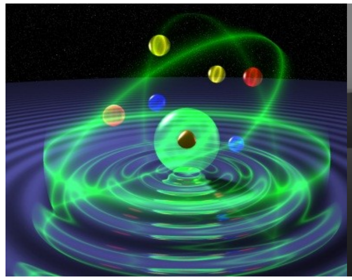
$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024 = 1\,K$
$2^{11} = 2048 = 2\,K$
$2^{12} = 4096 = 4\,K$
$2^{13} = 8192 = 8\,K$
$2^{14} = 16384 = 16\,K$
$2^{15} = 32768 = 32\,K$
$2^{16} = 65536 = 64\,K$
$2^{17} = 131072 = 128\,K$
$2^{18} = 262144 = 256\,K$
$2^{19} = 524288 = 512\,K$
$2^{20} = 1048576 = 1\,M$
$2^{21} = 2097152 = 2\,M$
$2^{22} = 4194304 = 4\,M$
$2^{23} = 8388608 = 8\,M$
$2^{24} = 16777216 = 16\,M$
$2^{25} = 33554432 = 32\,M$
$2^{26} = 67108864 = 64\,M$
$2^{27} = 134217728 = 128\,M$
$2^{28} = 268435456 = 256\,M$
$2^{29} = 536870912 = 512\,M$
$2^{30} = 1073741824 = 1\,G$
$2^{31} = 2147483648 = 2\,G$
$2^{32} = 4294967296 = 4\,G$

COMP557 Computer Graphics

- Fundamental mathematical, algorithmic and representational issues in computer graphics
- Learn by doing! 4 assignments.

Homogeneous Coordinates, 3D Affine Transformations, Ray Tracing, Rasterization, Z-buffer, Illumination Models, Perspective Projection, Anaglyphs, Mesh Data Structures, Curves, Surfaces, Subdivision, Simplification, Texture Mapping, Shadows, Radiosity, Colour, Compositing

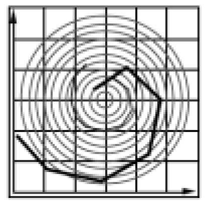




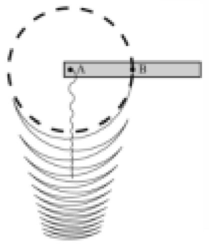


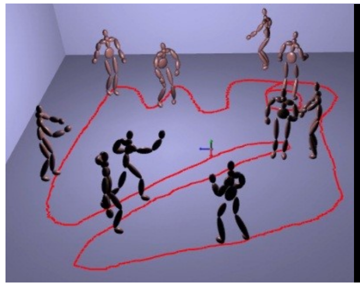
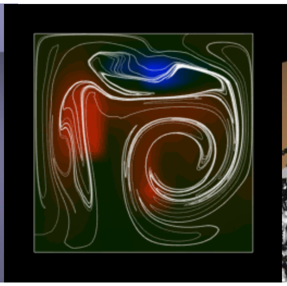
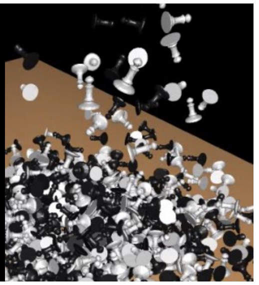
COMP559 Computer Animation

- Computational techniques for creating computer animation.
- Focus on physically based methods
- Learn with small interactive assignments and mini-project

ODEs, Numerical Integration, Stability, Mass-spring Systems, Constraints and Stabilization, Collision Detection, Collision Response, Contact LCPs, Motion Capture, Character Animation, Fluid Simulation (SPH and Eulerian), 3D Rigid Motion, IK, Elastic Solids

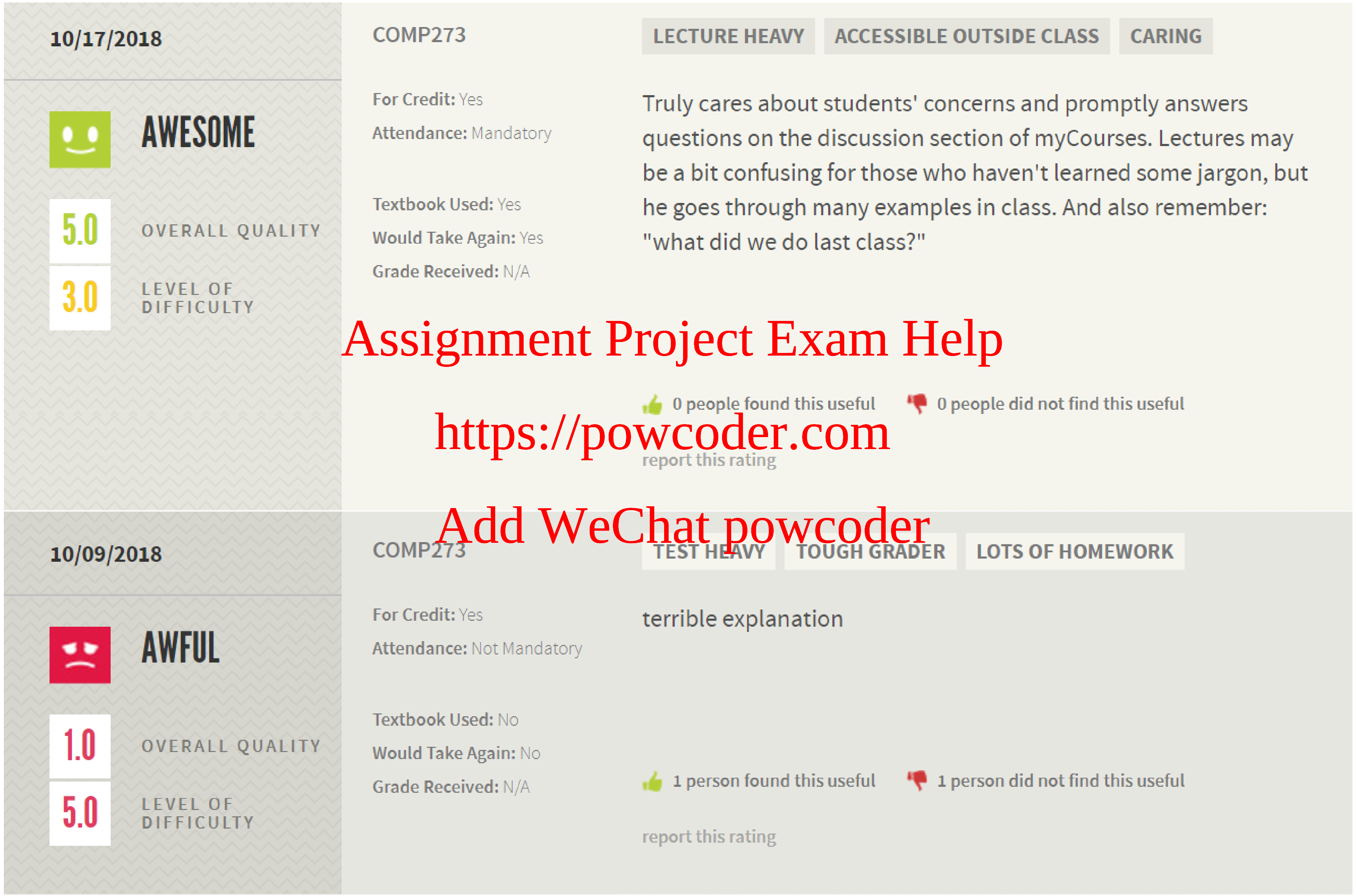




DON’T FORGET TO DO COURSE EVALUATIONS

…official evaluations, but also consider ratemyprofessors.com



10/17/2018

AWESOME

5.0 OVERALL QUALITY

3.0 LEVEL OF DIFFICULTY

COMP273

LECTURE HEAVY | ACCESSIBLE OUTSIDE CLASS | CARING

For Credit: Yes
Attendance: Mandatory

Textbook Used: Yes
Would Take Again: Yes
Grade Received: N/A

Truly cares about students' concerns and promptly answers questions on the discussion section of myCourses. Lectures may be a bit confusing for those who haven't learned some jargon, but he goes through many examples in class. And also remember: "what did we do last class?"

0 people found this useful | 0 people did not find this useful

report this rating

10/09/2018

AWFUL

1.0 OVERALL QUALITY

5.0 LEVEL OF DIFFICULTY

COMP273

TEST HEAVY | TOUGH GRADER | LOTS OF HOMEWORK

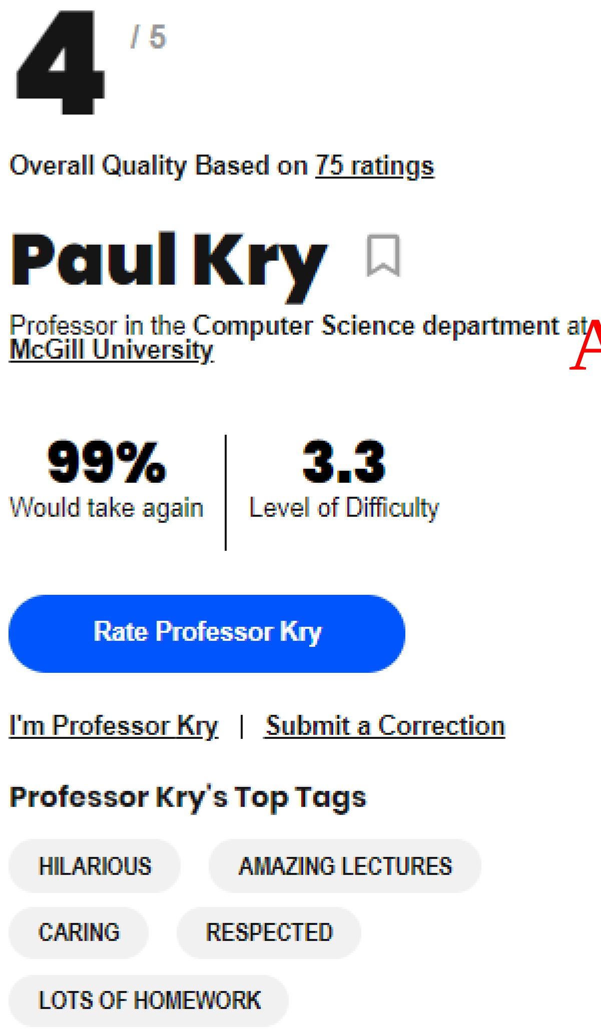
For Credit: Yes
Attendance: Not Mandatory

Textbook Used: No
Would Take Again: No
Grade Received: N/A

terrible explanation

1 person found this useful | 1 person did not find this useful

report this rating



4 / 5

Overall Quality Based on 75 ratings

Paul Kry

Professor in the Computer Science department at McGill University

99% Would take again

3.3 Level of Difficulty

Rate Professor Kry

I'm Professor Kry | Submit a Correction

Professor Kry's Top Tags

HILARIOUS

AMAZING LECTURES

CARING

RESPECTED

LOTS OF HOMEWORK

4 on Scale of 1 to 5? If 1 is 0% and 5 is 100%...

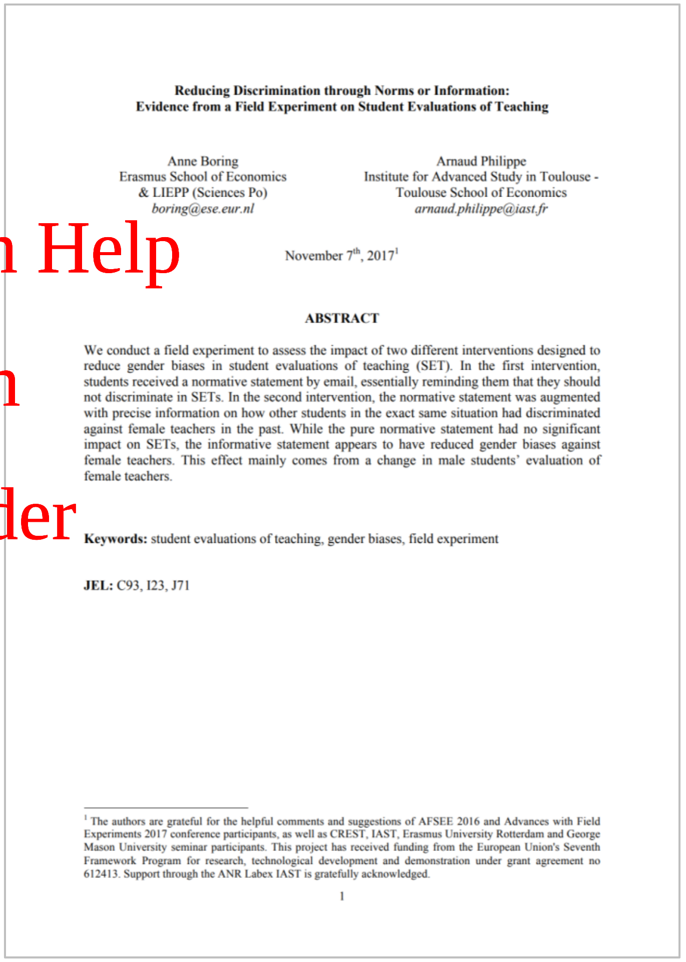
75% B+ **(just barely)**

Reducing discrimination through norms or information

[Boring, Philippe 2017]

a) Simply reminding people not to be biased when filling out their teaching evaluations seems not to have an effect.

b) If as well as the reminder, you inform people that that ***bias really does exist***, in their exact setting, then does help reduce the resulting bias.



Reducing Discrimination through Norms or Information:
Evidence from a Field Experiment on Student Evaluations of Teaching

Anne Boring
Erasmus School of Economics
& LIEPP (Sciences Po)
boring@ese.eur.nl

Arnaud Philippe
Institute for Advanced Study in Toulouse -
Toulouse School of Economics
arnaud.philippe@iast.fr

November 7th, 2017[1]

ABSTRACT

We conduct a field experiment to assess the impact of two different interventions designed to reduce gender biases in student evaluations of teaching (SET). In the first intervention, students received a normative statement by email, essentially reminding them that they should not discriminate in SETs. In the second intervention, the normative statement was augmented with precise information on how other students in the exact same situation had discriminated against female teachers in the past. While the pure normative statement had no significant impact on SETs, the informative statement appears to have reduced gender biases against female teachers. This effect mainly comes from a change in male students' evaluation of female teachers.

Keywords: student evaluations of teaching, gender biases, field experiment

JEL: C93, I23, J71

[1] The authors are grateful for the helpful comments and suggestions of AFSEE 2016 and Advances with Field Experiments 2017 conference participants, as well as CREST, IAST, Erasmus University Rotterdam and George Mason University seminar participants. This project has received funding from the European Union's Seventh Framework Program for research, technological development and demonstration under grant agreement no 612413. Support through the ANR Labex IAST is gratefully acknowledged.

1