THE HOLY BIBLE

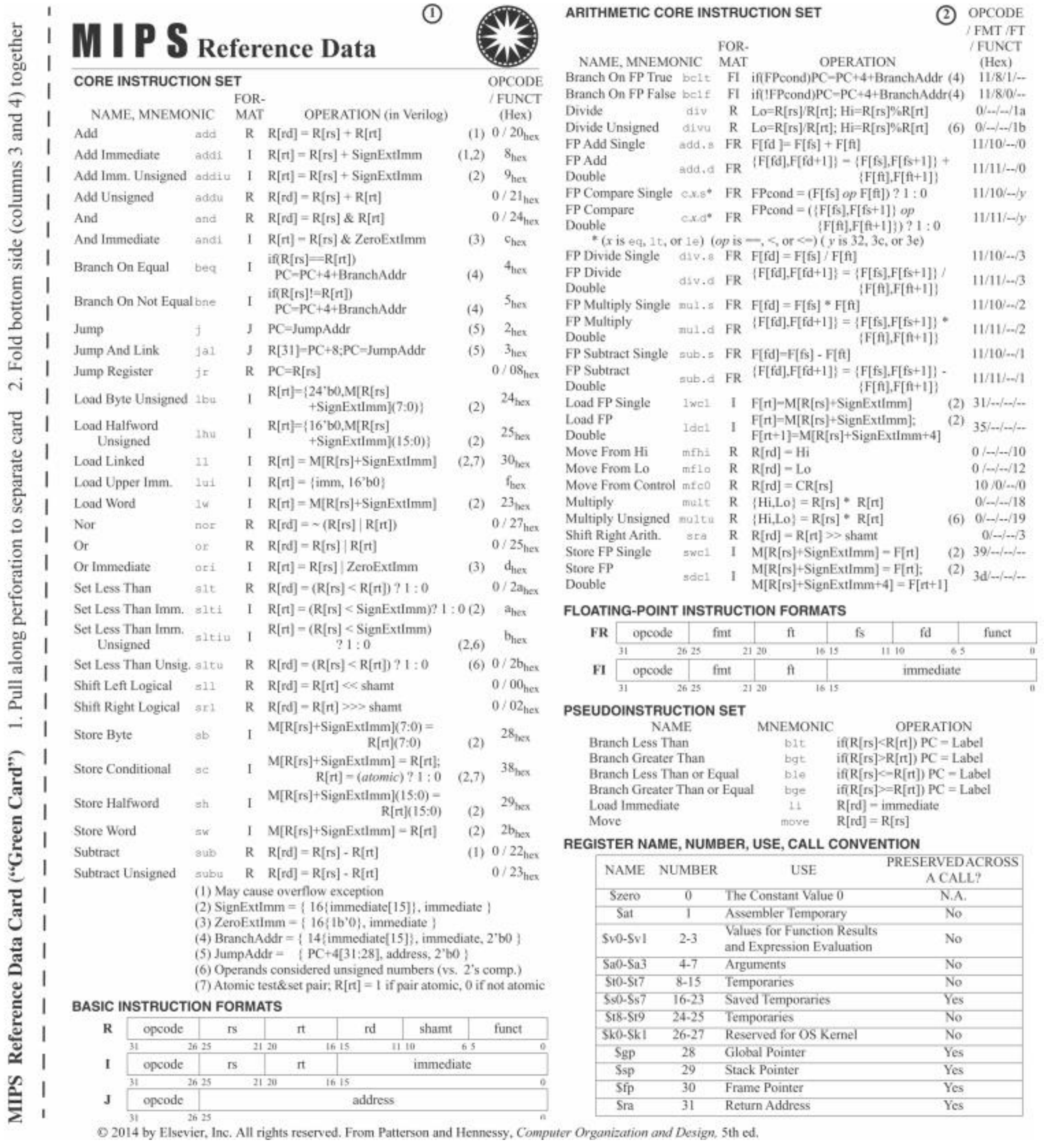
Computer Science 313

Compiled by Dan Richards.

Contents

MIPS

Instruction Reference



MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

①

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(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+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

ARITHMETIC CORE INSTRUCTION SET ②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
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/--
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
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
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 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 Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
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 Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
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 Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
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/--/--/--
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]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >> shamt		0/--/--/3
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/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		

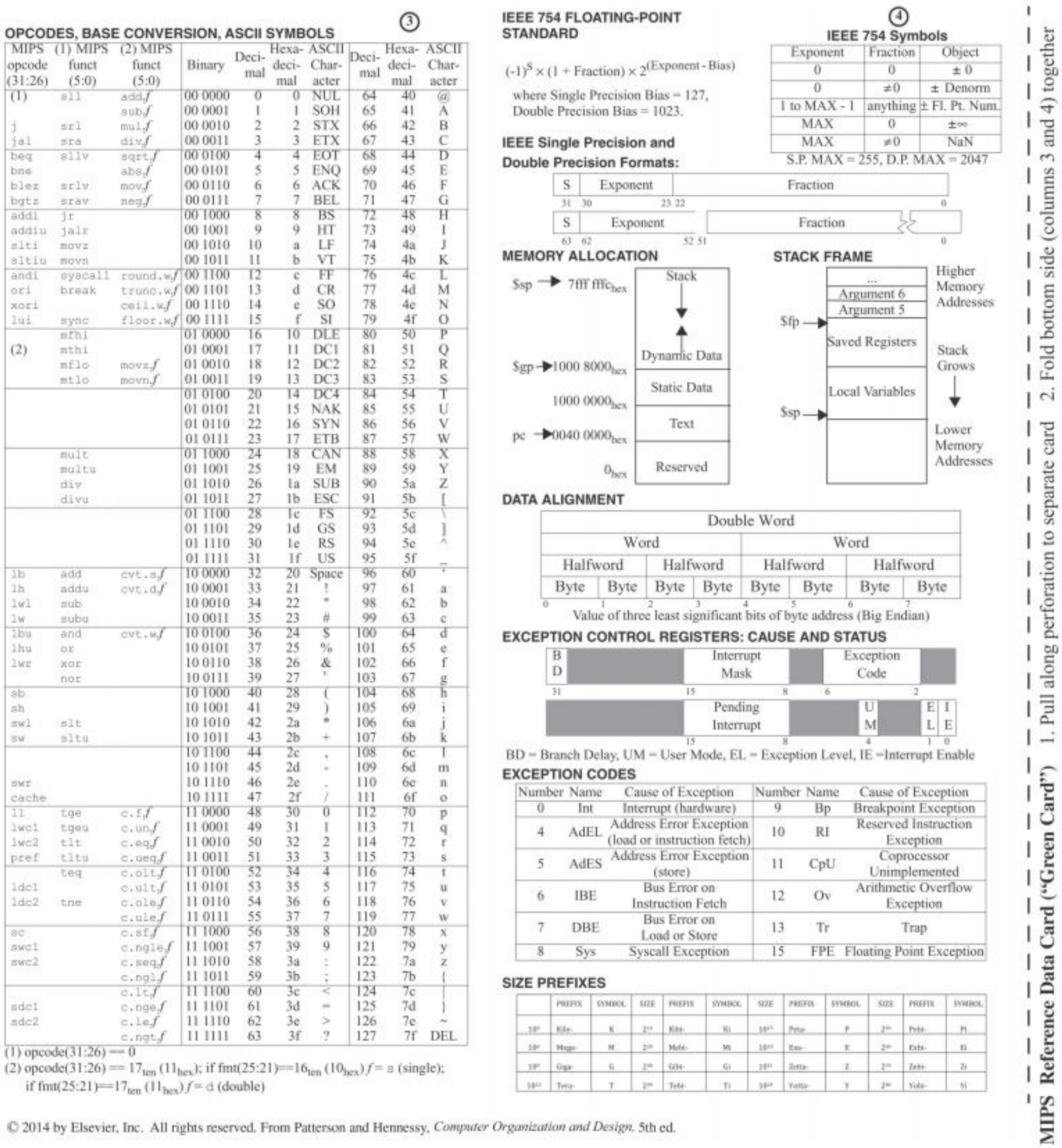
PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$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

Binary Reference



OPCODES, BASE CONVERSION, ASCII SYMBOLS

③

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)

IEEE 754 FLOATING-POINT STANDARD

④

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

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$_{hex}$

0$_{hex}$ — Reserved

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

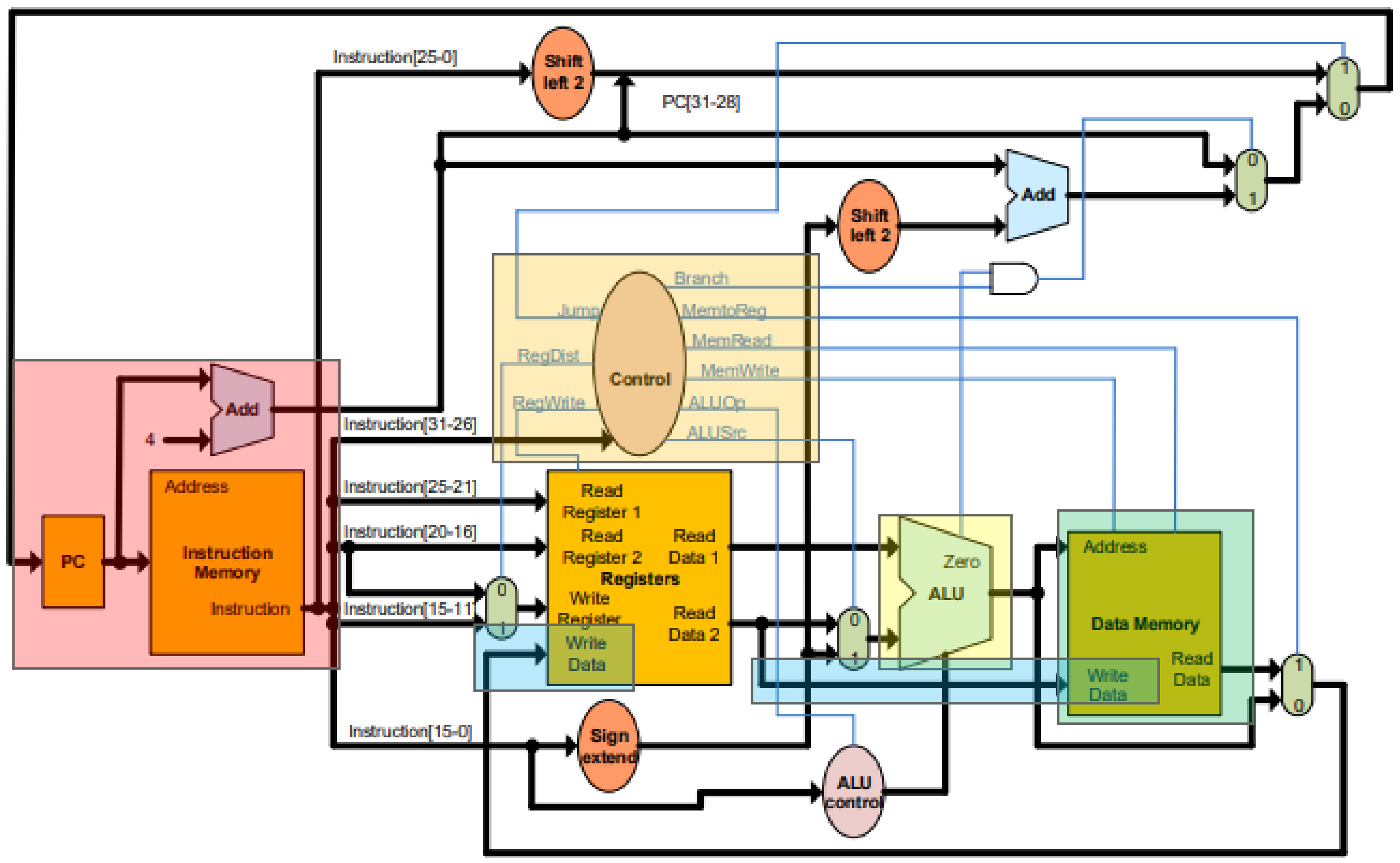
	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL
10^3	Kilo-	K	2^{10}	Kibi-	Ki	10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^6	Mega-	M	2^{20}	Mebi-	Mi	10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^9	Giga-	G	2^{30}	Gibi-	Gi	10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti	10^{24}	Yotta-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

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

Register Reference

Number	Name	Usage	Reserved across call	Binary	Hex	Decimal
0	$zero	Zero	N/A	0	0	0
1	$at	Assembler Temporary	No	1	1	1
2	$v0	Return values	No	10	2	2
3	$v1			11	3	3
4	$a0	Argument Values	No	100	4	4
5	$a1			101	5	5
6	$a2			110	6	6
7	$a3			111	7	7
8	$t0	Temporary	No	1000	8	8
9	$t1			1001	9	9
10	$t2			1010	A	10
11	$t3			1011	B	11
12	$t4			1100	C	12
13	$t5			1101	D	13
14	$t6			1110	E	14
15	$t7			1111	F	15
16	$s0	Saved	Yes	10000	10	16
17	$s1			10001	11	17
18	$s2			10010	12	18
19	$s3			10011	13	19
20	$s4			10100	14	20
21	$s5			10101	15	21
22	$s6			10110	16	22
23	$s7			10111	17	23
24	$t8	Temporary	No	11000	18	24
25	$t9			11001	19	25
26	$k0	Kernel use	No	11010	1A	26
27	$k1			11011	1B	27
28	$gp	Global Pointer	Yes	11100	1C	28
29	$sp	Stack Pointer	Yes	11101	1D	29
30	$fp	Frame Pointer	Yes	11110	1E	30
31	$ra	Return Address	Yes	11111	1F	31

Single Cycle Processor Diagram - Annotated



Processing Stages

1 – Instruction Fetching 🟥

Also increases PC by 4

2 – Instruction Decoding 🟧

Determine the Control signals

3 – ALU Operation 🟨

Calculate address / data

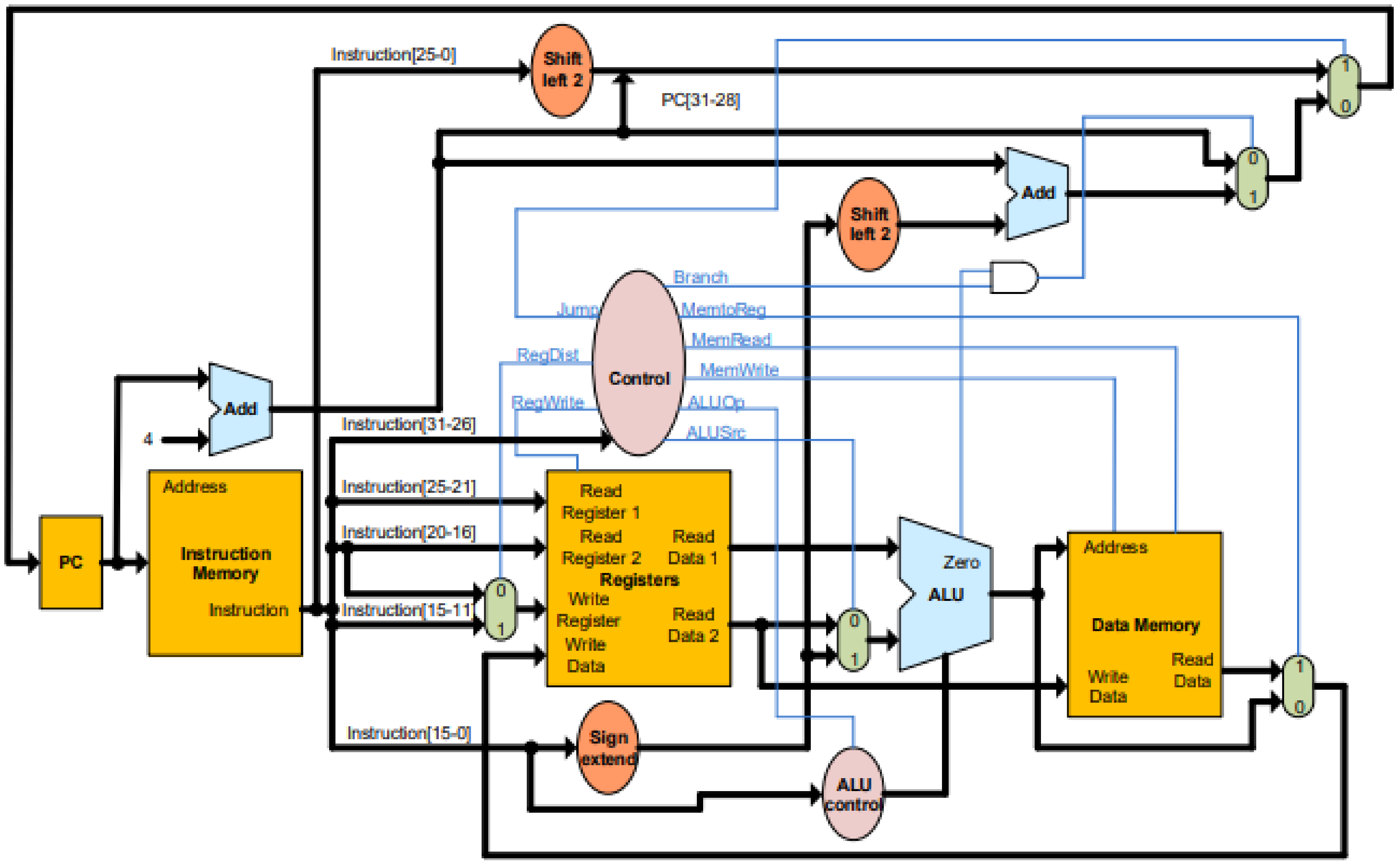
4 – Memory Access 🟩

Read data from memory

5 – Write back 🟦

Store the result into mem/register

Single Cycle Processor Diagram



Single Cycle Processor – Performance and overview

Consider: $\boldsymbol{N}\ x\ \boldsymbol{CPI}\ x\ \left(\frac{1}{\boldsymbol{f}}\right) = \boldsymbol{N}\ x\ \boldsymbol{CPI}\ x\ \boldsymbol{T}\ Seconds = Performance$

Reduce N:

1. Make instructions that do more. – CISC (ISA)
2. Use better compilers

Use less cycles to perform instructions (CPI):

1. Simpler instruction – RISC
2. Use multiple ALU / cores in parallel – process architecture

Increase Clock Frequency:

1. Find a newer technology – semiconductor
2. Redesign time critical components
3. Adopt Pipelining

Strengths:

1. Simple
2. easy to design
3. less power
4. lower manufacturing cost

Weakness:

1. Cycle time limited by (lw)
2. Duplicate components, 2 Adders / ALU and 2 memories

Multi Cycle Processor – Performance and overview

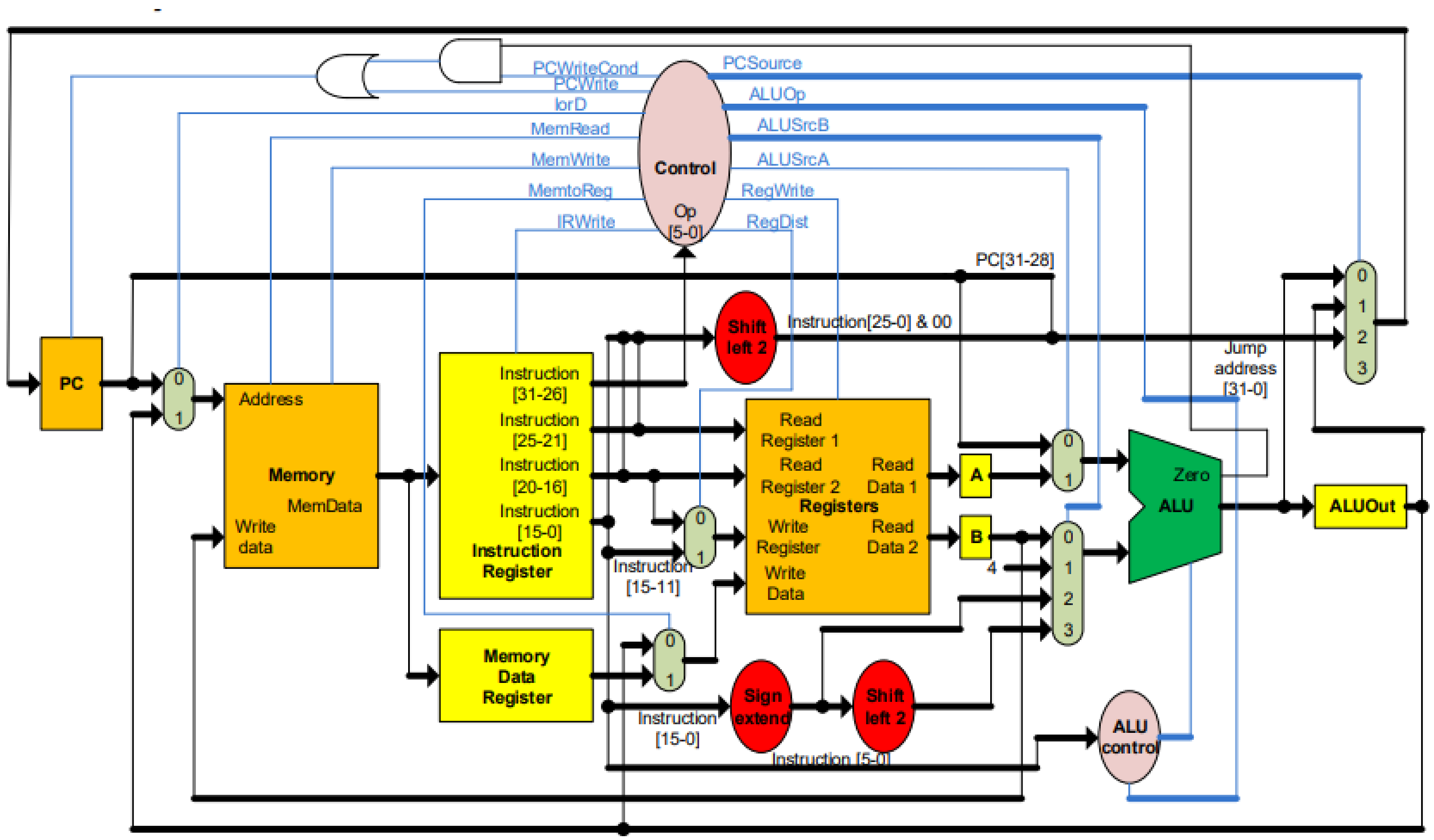
Strengths:

1. Higher clock speed (***f***)
2. Simplier sintructions run faster (better CPI)
3. Reuse expensive hardware on multiple cycles (More simplistic)
4. Smaller in size due to 1 MEM and 1 ALU
5. Common case is faster due to not waiting for LW or critical path

Weaknesses:

1. More Complex
2. Sequencing overhead paid many times (power)

Multi Cycle Processor Diagram



Main Control Unit Signals (Single Cycle)

Signal Name	Effect when de-asserted	Effect when asserted
RegDst	The register destination number for the "Write register" comes from the "rt" field	The register destination number for the "Write register" comes from the "rd" field
RegWrite	None	The register on the "Write register" input is written with the value on the "Write data" input
ALUSrc	The second ALU operand comes from the second register file output ("Read data 2")	The second ALU operand is the sign-extended, lower 16 bits of the instruction
MemRead	None	Data memory contents designated by the address input are put on the "Read data" output
MemWrite	None	Data momory contents designated by theaddress input are replaced by the value on the "Write data" input
MemtoReg	The value fed to the register "Write data" input comes from the ALU	The value fed to the register "Write data" input comes from the data memory
Branch	The PC is replaced by the output of the adder that computes the value of PC+4	The PC is replaced by the output of the adder that computes the branch target
Jump	The PC is replaced by the output controlled by Branch	The PC is replaced by jump target

Main Control Unit – Instruction Lookup (Single Cycle)

Signal name	R-format	lw	sw	beq	bne	j	addi
Instruction [31-26]	0x00	0x23	0x2B	0x04	0x05	0x02	0x08
RegDst	1	0	X	X	X	X	0
ALUSrc	0	1	1	0	0	X	1
MemtoReg	0	1	X	X	X	X	0
RegWrite	1	1	0	0	0	0	1
MemRead	0	1	0	0	0	0	0
MemWrite	0	0	1	0	0	0	0
Branch	0	0	0	1	1	0	0
ALUOp	010	000	000	001	111	000	011
Jump	0	0	0	0	0	1	0

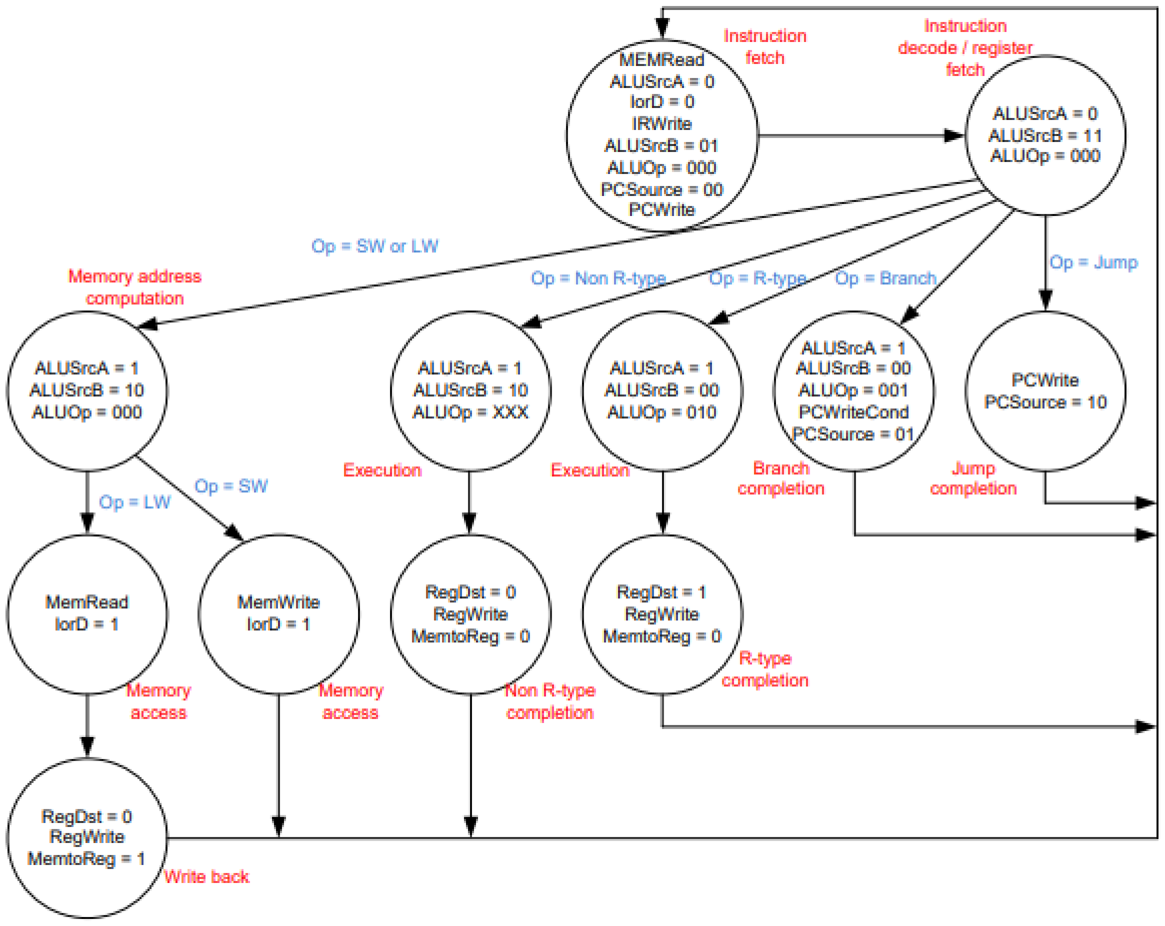
Control Unit Signals (Multi Cycle)

Signal name	Effect when de-asserted	Effect when asserted
RegDst	The register file destination number of the "Write regsiter" come from the rt field	The register file destination number for the "Write register" comes from the rd field
RegWrite	None	The general-purpose register selected by the "Write register" number is written with the value of the "Write data" input
ALUSrcA	The first ALU operand is the PC	The first ALU operand comes from the A register
MemRead	None	Content of memory at the location specified by the "Address" input is put on "Memory data" output
MemWrite	None	Memory contents at the location specified by the "Address" input is replaced by the value on the "Write data" input
MemtoReg	The value fed to the register file "Write data" input comes from ALUOut	The value fed to the register file "Write data" input comes from the MDR
IorD	The PC is used to supply the address to the memory unit	ALUOut is used to supply the address to the memory unit
IRWrite	None	The output of the memory is written into the IR
PCWrite	None	The PC is written and the source is controlled by PCSource
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active

Control Unit Signals to ALU(Multi Cycle)

Signal name	Value	Effect
ALUOp	000, 011	The ALU performs an add operation
	001	The ALU performs a subtract operation
	010	The functional field of the instruction determines the ALU instruction
	100	The ALU performs an and operation
	101	The ALU performs an or operation
	110	The ALU performs an xor operation
	111	The ALU performs a subtract operation but have negative output of Zero
ALUSrcB	00	The second input to the ALU comes from the B register
	01	The second input to the ALU is constant 4
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left by 2 bits
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing
	10	The jump target address (IR[25-0]) shifted left 2 bits and concatenated with PC + 4[31-28] is sent to PC for writing

Finite State Diagram (Multi Cycle)



Processor Performance Equations

	IC	CPI	Clock Rate
Program	X		
Compiler	X	X	
ISA	X	X	
Organisation		X	X
Technology			X
Algorithm	X	~	
Programming Language	X	X	
Compiler	X	X	
ISA	X	X	
Hardware		X	X

Power

$$Dynamic\ Power = 0.5 * capacitive\ load * Voltage^2 * Frequency$$

Look at lecture 5 for the other equation….

Processing Time

$$CPU\ Time\ =\ Clock\ Cycles\ \cdot\ Cycle\ Time$$

$$CPU\ Time = \frac{Cycles}{Clock\ Rate}$$

$$Clock\ Cycles = Instruction\ Count * Cycles\ Per\ Instruction$$

$$CPU\ Time = \frac{IC * CPI}{Clock\ Rate} =\ Instruction\ Count * CPI * Clock\ Cycle\ Time$$

Division of seconds	Seconds
1 millisecond	0.001
1 microsecond	0.000,001
1 nanosecond	0.000,000,001
1 picosecond	0.000,000,000,001

Relative performance

"X is n times faster than Y" or "How much faster is X to Y?"

$$\text{Performance}_X / \text{Performance}_Y = \text{Execution Time}_Y / \text{Execution Time}_X$$

Floating Point arithmetic

IEEE 754 Floating Point Standard

Single Precision

S	Exponent (8 bits)	Fraction (23 bits)

Exponent bias: 127

Double Precision

S	Exponent (11 bits)	Fraction (52 bits)

Exponent bias: 1203

Decimal to FP Conversion

1. **Find the sign**

0	Positive
1	Negative

2. **Find the fraction(45.45)**

2.1 Find the binary value of the integer component.

45 = 101101

2.2 Find the binary value of the decimal component.

0.45 = 011100110011001100……..

0.45 x 2	=	0	.90	
0.90 x 2	=	1	.80	
0.80 x 2	=	1	.60	Repeating Sequence
0.60 x 2	=	1	.20	
0.20 x 2	=	0	.40	
0.40 x 2	=	0	.80	
0.80 x 2	=	1	.60	

2.3 **Add the values together**

101101.0111001100110011001100 (Until satisfies the 23-bit slot)

2.4 Find the exponent of 2 to make the decimal before the leading 1.

.101101.0111001100110011001100

$\longleftarrow$

5 places

2.5 **Add 127 to this exponent and convert to binary**

5 + 127 = 132

10000100

2.6 Combine the binary strings

0	10000100	10110101110011001100110
S	Exponent (8 bits)	Fraction (23 bits)

De-normal Numbers

I don't quite understand this…

Infinity and NaN (Not a Number)

Infinity

Sign	Exponent	Fraction	Value
1	11111111	000000000000……	$-\infty$
0	11111111	000000000000……	$+\infty$

Not a Number

Sign	Exponent	Fraction	Value
1/0	11111111	Not equal 000000……	*NaN*

Notes on Floating Point Numbers

1. Floating Point is associative which means that doing things in different order matters
2. Floating Point numbers have limited precision
3. Floating point only gives approximations of real results

Adding

1. Ensure the signs are aligned
2. Perform the calculation

Multiple

1. Change the exponent to that the decimal is removed
   a. <- = +
   b. -> = -

2. Multiple the two binary numbers together
3. Put the exponent back

$$\begin{aligned}
A \times B \quad &= 1.1111 \times 2^5 \times 1.0011 \times 2^{14} \\
&= 11111 \times 2^1 \times 10011 \times 2^{10} \\
&= 1001001101 \times 2^{11} \\
&= 1.0010 \times 2^{20} \\
A \times C \quad &= 1.1111 \times 2^5 \times 1.0010 \times 2^{14} \\
&= 11111 \times 2^1 \times 10010 \times 2^{10} \\
&= 1000101110 \times 2^{11} \\
&= 1.0001 \times 2^{20} \\
(A \times B) - (A \times C) \quad &= 1.0010 \times 2^{20} - 1.0001 \times 2^{20} \\
&= 0.0001 \times 2^{20} \\
&= 1 \times 2^{16}
\end{aligned}$$

$$\begin{aligned}
B - C \quad &= 1.0011 \times 2^{14} - 1.0010 \times 2^{14} \\
&= 0.0001 \times 2^{14} \\
A \times (B - C) \quad &= 1.1111 \times 2^5 \times 0.0001 \times 2^{14} \\
&= 1.1111 \times 2^5 \times 1 \times 2^{10} \\
&= 1.1111 \times 2^{15}
\end{aligned}$$

Subtraction

1. Get the exponents of the binary aligned.
   a. Does one of the binary values make it equal 1?
   b. What exponent is easier to change
2. Perform the subtraction

Division

1. Follow the multiplication steps.

Integer arithmetic

Addition Overflow Conditions

(+) + (+), result has 1 as LSB, then overflow.

(-) + (-) result has 0 as LSB then overflow

(+) + (-) overflow is not possible.

Subtraction Overflow Conditions

(+) – (+) no overflow

(-) - (-) no overflow

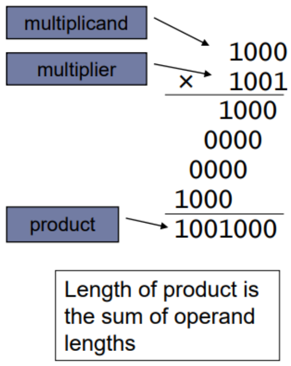
(-) – (+), overflow if MSB is 0 (or positive).

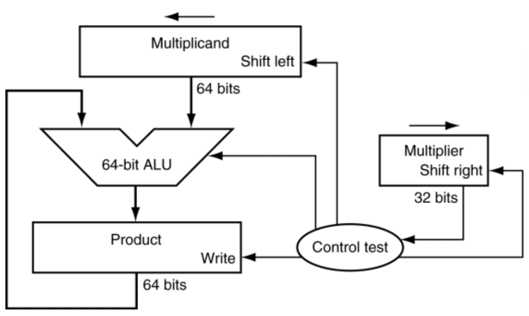
Remember naturally if subtracting a positive number from a negative number should yield a more negative number. If you've overflown the bounds of the value, then the number will become positive.

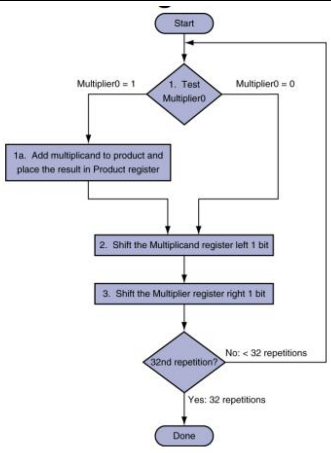
(+) – (-), overflow is MSB is 1 (or negative).

Remember naturally if subtracting a negative number from a positive number it should yield a more positive number. If you've overflown the bounds of the value, then the number will become negative.

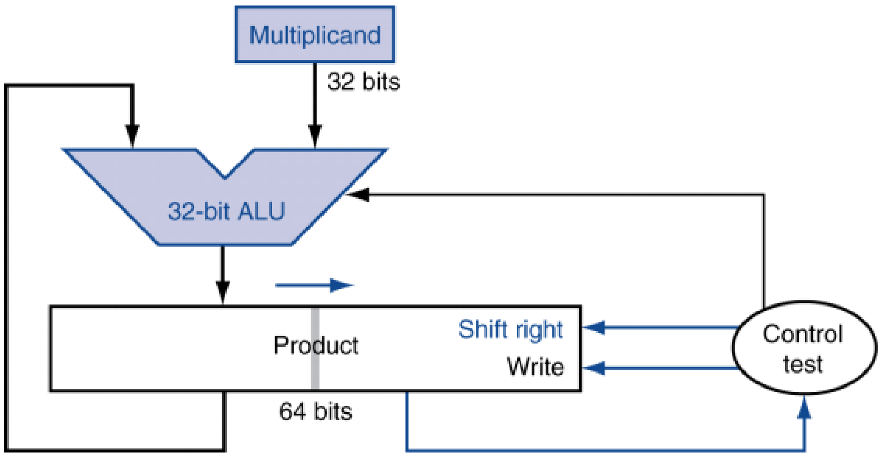
Multiplication






Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Multiplication Optimised



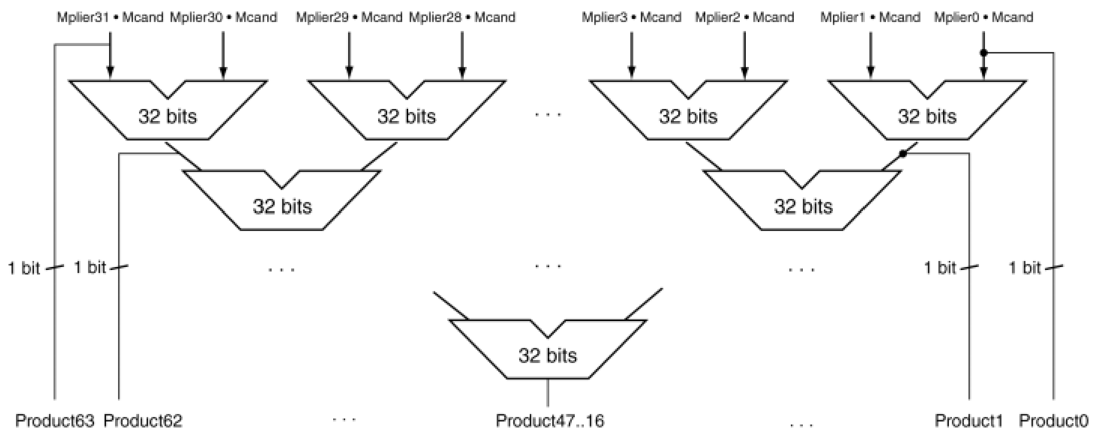
Comments:

The product register holds both the product and the multiplier at the same time. The higher 32 bits hold the product with the lower 32 bits holding the multiplier. The control test gets the LSB (the first bit of the multiplier) checks to see if the product needs to be added with the multiplicand or all 0s. The multiplicand or all 0s are only added to the highest 32bits, which makes sense for multiplication.

The product value is initially at the MSB filling 32bits in the product register and is shifted right to add the multiplicand to the left most 32bits. *This means that it takes 32 clock cycles to calculate the multiplier.*

Super-Duper Optimised Multiplication

Comments: Don't need to understand, but with logic you can add the different sections of the multiplier and the multiplicand together using multiple 32-bit adders which select the bits which they add.



Division