CS2100 Computer Organisation

AY2021/22 Semester 1

1. C Programming Language

1.1. Pointer

- `&a`: Address of `a`.
- `%p`: Format specifier for addresses.
- `int *a_ptr;`: Declaring a pointer variable.
- `a_ptr = &a;` or `int *a_ptr = &a;`: Assigning value to a pointer.
- `*a_ptr` is synonymous with `a`.

1.2. Array

- `a` is synonymous with `&a[0]`.
- An array that ends with `\0` is a string.

1.3. Structure

- We can directly assign by the name.
- Structure is similar to variable.
- `(*a_ptr).b` is equivalent to `a_ptr->b`.

2. Data Representation and Number Systems

2.1. Weighted-Positional Number System

- $(a_n a_{n-1} \dots a_1 a_0 . f_1 f_2 \dots f_m)_R$
  $= a_n R^n + a_{n-1} R^{n-1} + \dots + a_1 R + a_0 + f_1 R^{-1} + f_2 R^{-2} + \dots + f_m R^{-m}$
- Base-R to Decimal Conversion:

Division-by-2

	10	remainder	
/2	5	0	← LSB
/2	2	1	
/2	1	0	
/2	0	1	← MSB
$(1010)_2$			

Multiplication-by-2

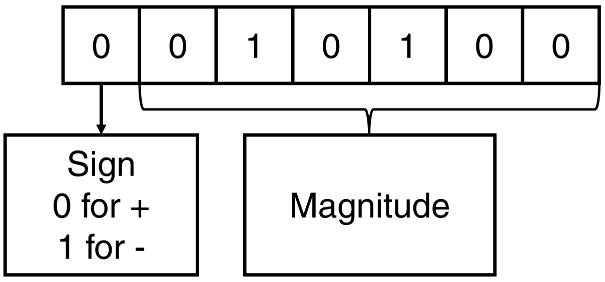
	0.3125	carry	
*2	0.625	0	← MSB
*2	0.25	1	
*2	0.5	0	
*2	0	1	← LSB
$(0.0101)_2$			

2.2. ASCII Table

Refer to Appendix A.

2.3. Signed Binary Numbers

- Sign-and-Magnitude:

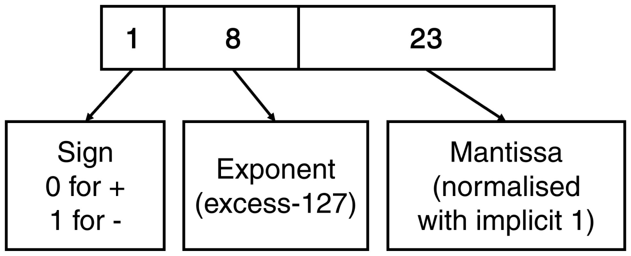


- 1s Complement
  $-x = 2^n - x - 1$ (negate all the bits)
  - If there is a carry-out of MSB during addition, add 1 to the result.
- 2s Complement
  $-x = 2^n - x$ (negate all the bits, then add 1)

2.4. Excess Representation

- Excess-n Representation: Subtract n from every number

2.5. Floating Point Numbers



3. MIPS

3.1. Instruction Formats

- R-Formats

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

- I-Formats

6	5	5	16
opcode	rs	rt	immediate

- J-Formats

6	26
opcode	target address

  - First 4 bits of PC + target address + 00

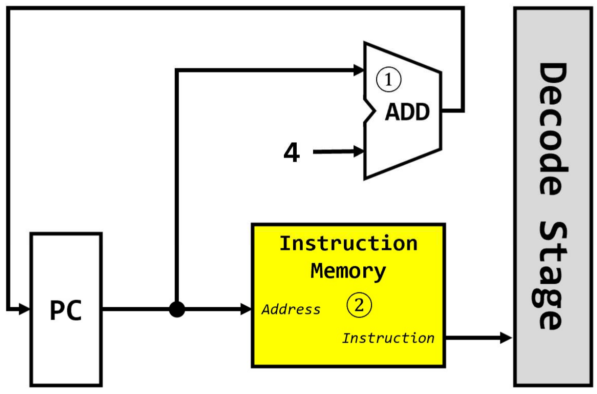
3.2. MIPS Reference Page

Refer to Appendix B.

4. Datapath and Control

4.1. Instruction Execution Cycle

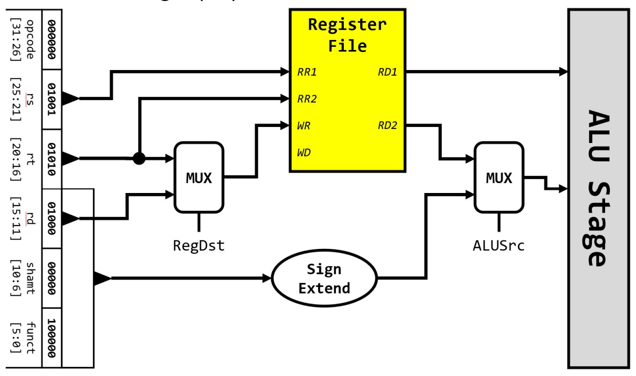
- Fetch Stage (IF)



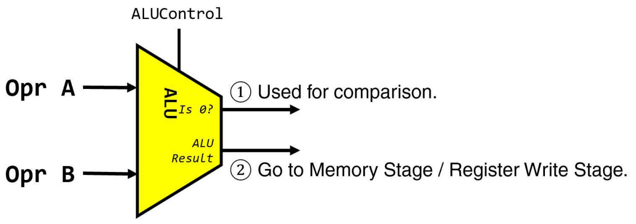
① Fetch instruction from PC address.

② Increment PC by 4.

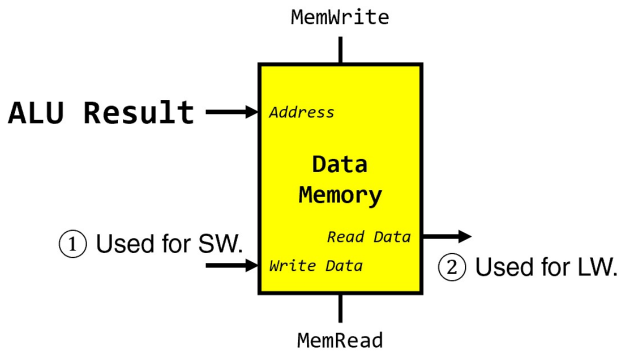
- Decode Stage (ID)



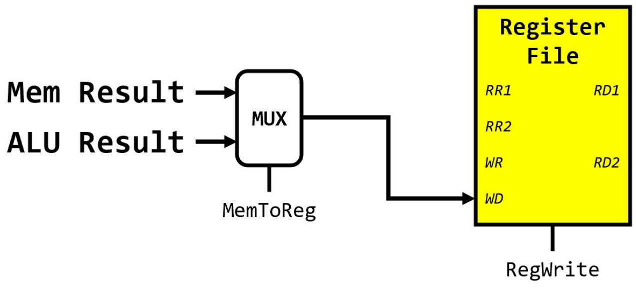
- ALU Stage (EX)



- Memory Stage (MEM)



- Register Write Stage (WB)



4.2. Complete Datapath

Refer to Appendix C.

4.3. Control

- Control Signals

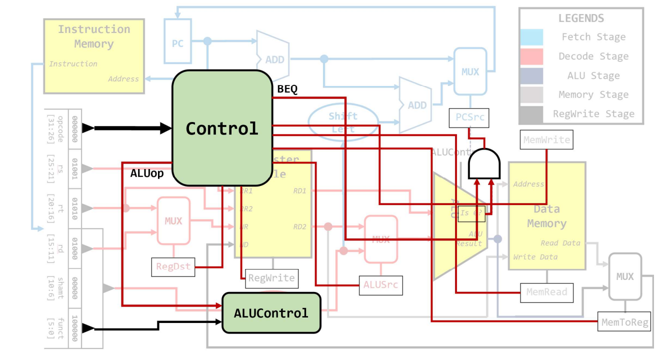
RegDst	Select destination register
RegWrite	Enable writing of register
ALUSrc	Select 2nd ALU operand
ALUControl	Select ALU operation
MemRead	Enable reading of data memory
MemWrite	Enable writing of data memory
MemToReg	Select data to write register
PCSrc	Select next PC value

- ALUControl Signal

ALUControl	Function
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	SLT
1100	NOR

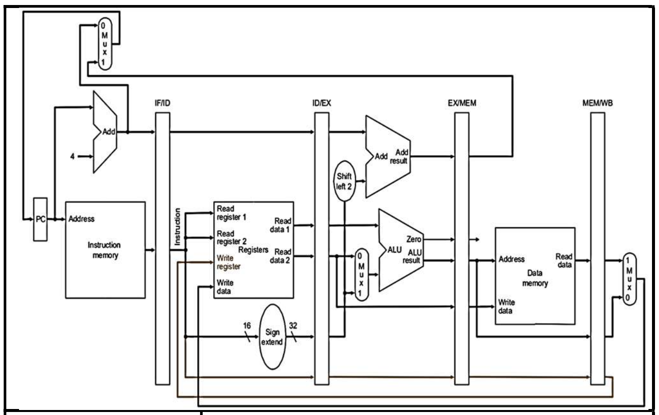
  - ALUControl signal is generated from 2-bit ALUop signal (LW/SW – 00; BEQ – 01; R-type – 10) and optionally 6-bit Funct field.

- Implementation



4.4. Pipelining

- Pipeline Datapath



Pipeline Register	What data does this register receive?	What data does this register supply?
IF/ID	(1) Instruction (2) PC + 4	(1) Read register 1 & 2 (2) 16-bit offset
ID/EX	(1) Read data 1 & 2 (2) 32-bit immediate value (3) PC + 4	(1) Read data 1 & 2 (2) 32-bit immediate value (3) PC + 4
EX/MEM	(1) (PC + 4) + (immediate × 4) (2) ALU result (3) isZero? signal (4) Read data 2	(1) (PC + 4) + (immediate × 4) (2) ALU result (3) isZero? signal

	(4) Read data 2
MEM/WB	(1) ALU result (2) Memory read data
	(1) ALU result (2) Memory read data

- Cycle Time and Execution Time
  - Single-cycle processor:

$$CT = \sum_{k=1}^{N} T_k$$

(*CT*: Cycle time; *N*: Number of stages; T_k: Time for operations in stage *k*)

$$ET = I \times CT$$

(*ET*: Execution time; *I*: Number of instructions)

  - Multi-cycle processor:

$$CT = \max(T_k)$$
$$ET = I \times \overline{CPI} \times CT$$

($\overline{CPI}$: Average cycles per instruction)

  - Pipeline processor:

$$CT = \max(T_k) + T_d$$

(T_d: Overhead for pipelining)

$$ET = (I + N - 1) \times CT$$

- Number of Cycles
  - Ideal case:

$$I + N - 1$$

(*I*: Number of instructions)

  - Without data forwarding:

Instruction	Delay Caused
RAW	+2
Load Word	+2
Branch at MEM	+3
Branch at ID	+1

  - With data forwarding:

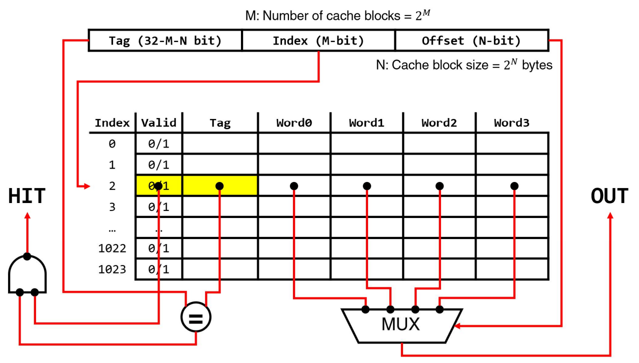
Instruction	Delay Caused
RAW	+0
Load Word	+1
Branch at MEM	+3
Branch at ID	+1
RAW + Branch at ID	+1
Load + Branch at ID	+2

4.5. Cache

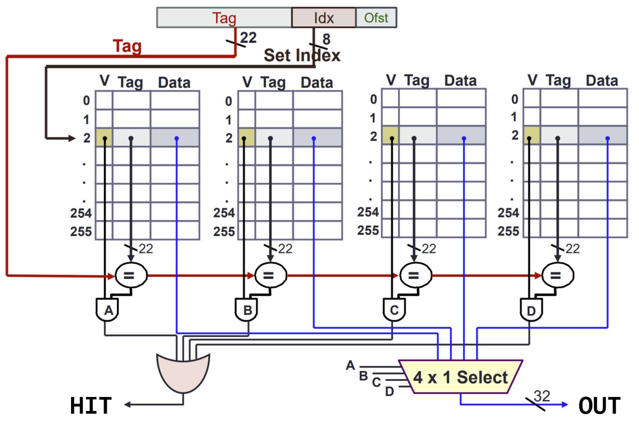
- $\overline{T_a} = P_{hit} T_{hi} + (1 - P_{hit}) T_{miss}$

($\overline{T_a}$: Average access time; P_{hi} : Hit rate; T_{hi} : Hit time; T_{miss}: Miss penalty)

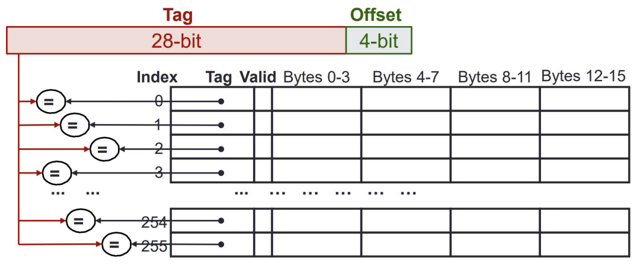
- Direct Mapped Cache



- *n*-way Set Associative Cache



- Fully Associative Cache



- Block Replacement Policy
  - Least Recently Used
  - First in First out
  - Random Replacement
  - Least Frequently Used

5. Circuits

5.1. Boolean Algebra

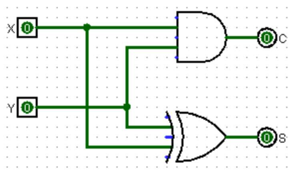
- Laws of Boolean Algebra

Identity Laws	
A + 0 = 0 + A = A	A · 1 = 1 · A = A
Inverse/Complement Laws	
A + A' = A' + A = 1	A · A' = A' · A = 0
Commutative Laws	
A + B = B + A	A · B = B · A
Associative Laws	
A + (B + C) = (A + B) + C	A · (B · C) = (A · B) · C
Distributive Laws	
A · (B + C) = A · B + A · C	A + (B · C) = (A + B) · (A + C)
Idempotency	
X + X = X	X · X = X
One/Zero Element	
X + 1 = 1 + X = 1	X · 0 = 0 · X = 0
Involution	

(X')' = X	
Absorption 1	
X + X · Y = X	X · (X + Y) = X
Absorption 2	
X + X' · Y = X + Y	X · (X' + Y) = X · Y
De Morgan's	
(X + Y)' = X' · Y'	(X · Y)' = X' + Y'
Consensus	
X · Y + X' · Z + Y · Z = X · Y + X' · Z	(X + Y) · (X' + Z) · (Y + Z) = (X + Y) · (X' + Z)

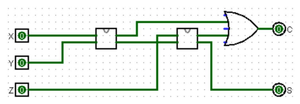
5.2. Adder

- Half-Adder



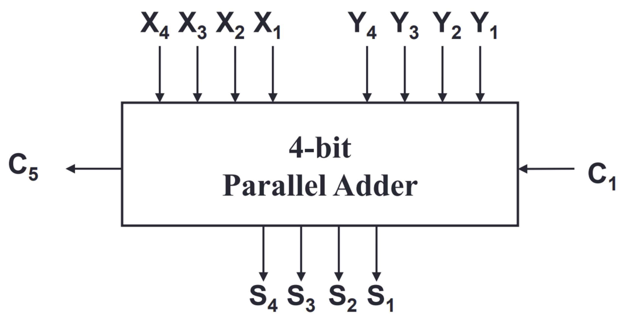
Input		Output	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Full Adder



Input			Output	
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- 4-bit Parallel Adder



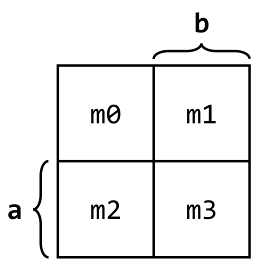
$$X_4X_3X_2X_1 + Y_4Y_3Y_2Y_1 + C_1 = C_5S_4S_3S_2S_1$$

5.3. K-Map

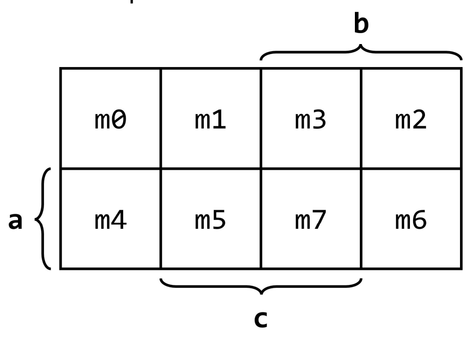
- Gray Code: Only a single bit changes from one code value to the next.

Decimal	Binary	Gray Code	Decimal	Binary	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

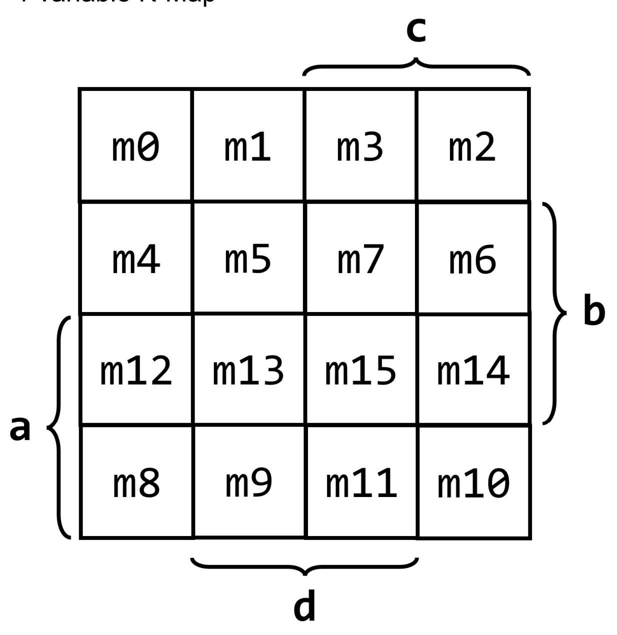
- 2-variable K-Map



- 3-variable K-Map



- 4-variable K-Map

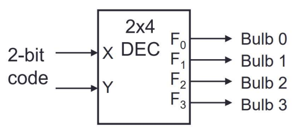


- Prime Implicant: A product term obtained by combining the maximum possible number of minterms from adjacent squares in the map.

- Essential Prime Implicant: A prime implicant that includes at least one minterm that is not covered by any other prime implicant.

- If $\alpha \neq \beta$, then:
  - $\mathrm{m}\alpha \cdot \mathrm{m}\beta = 0$
  - $\mathrm{M}\alpha + \mathrm{M}\beta = 1$

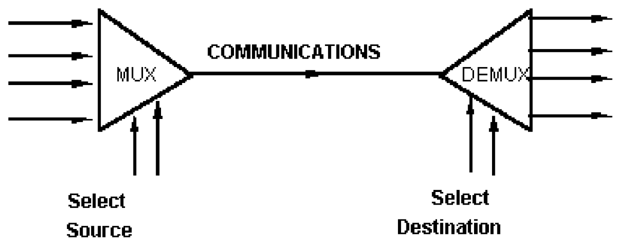
5.4. Decoder and Encoder



Input		Output			
X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

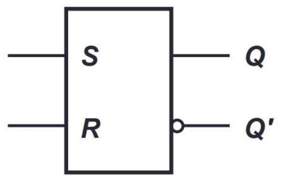
- Encode is reverse of decoder.

5.5. Multiplexer and Demultiplexer



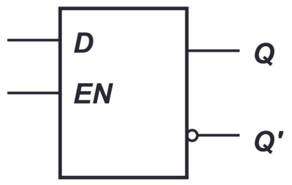
5.6. Latch

- Active-High S-R Latch



S	R	Q	Q’	State
0	0	Q	Q’	NC
0	1	1	0	Set
1	0	0	1	Reset
1	1	0	0	Invalid

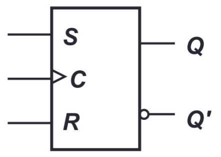
- Gated D Latch



EN	D	Q	State
1	0	0	Reset
1	1	1	Set
0	X	Q	NC

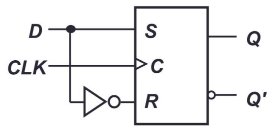
5.7. Flip-flop

- S-R Flip-flop



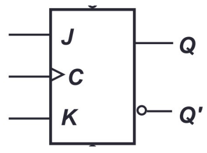
S	R	CLK	Q(*t*+1)	State
0	0	X	Q(*t*)	NC
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	Invalid

- D Flip-flop



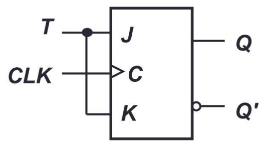
D	CLK	Q(*t*+1)	State
0	↑	0	Reset
1	↑	1	Set

- J-K Flip-flop



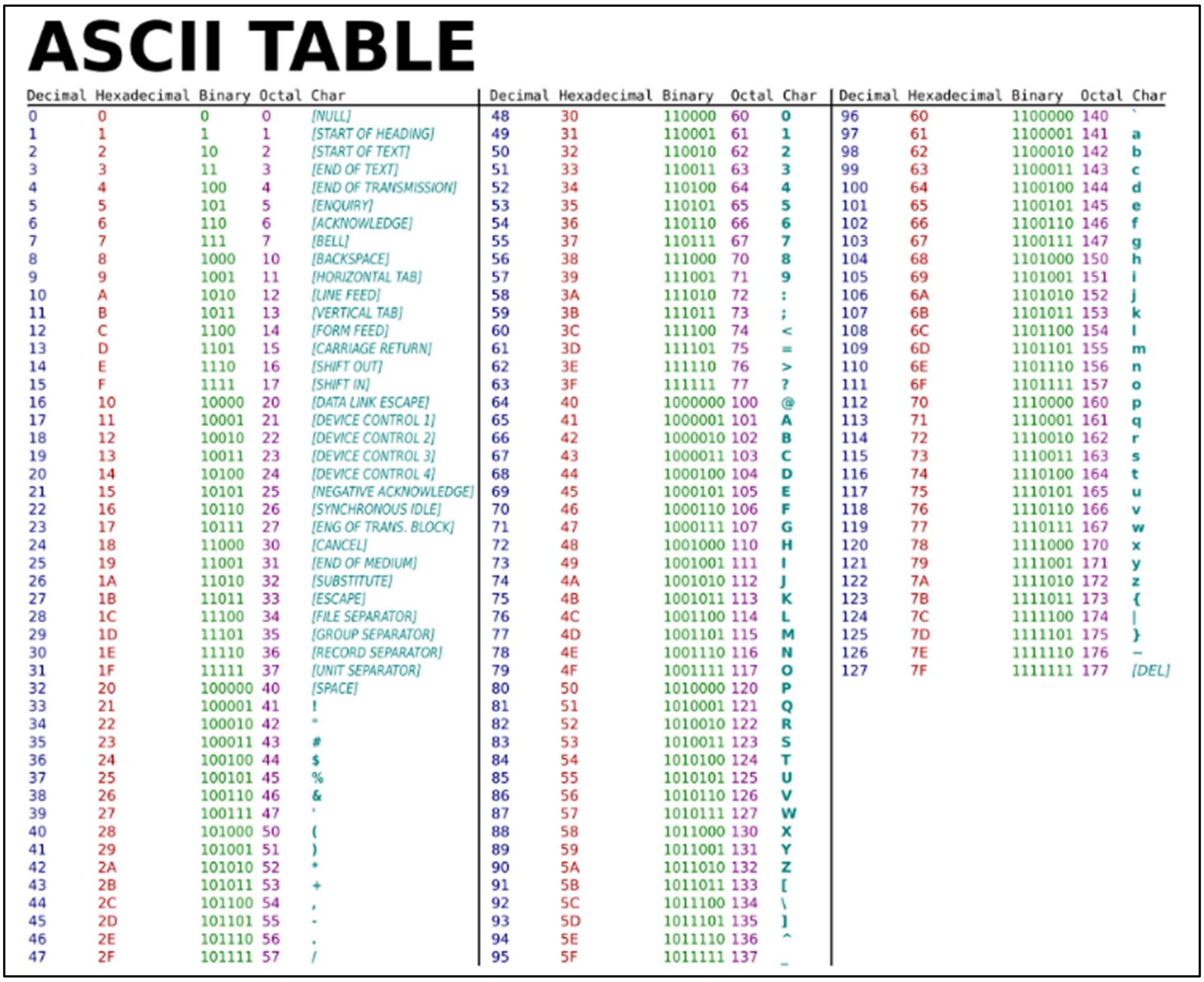
J	K	CLK	Q(*t*+1)	State
0	0	↑	Q(*t*)	NC
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q(*t*)’	Toggle

- T Flip-flop



D	CLK	Q(*t*+1)	State
0	↑	Q(*t*)	NC
1	↑	Q(*t*)’	Toggle

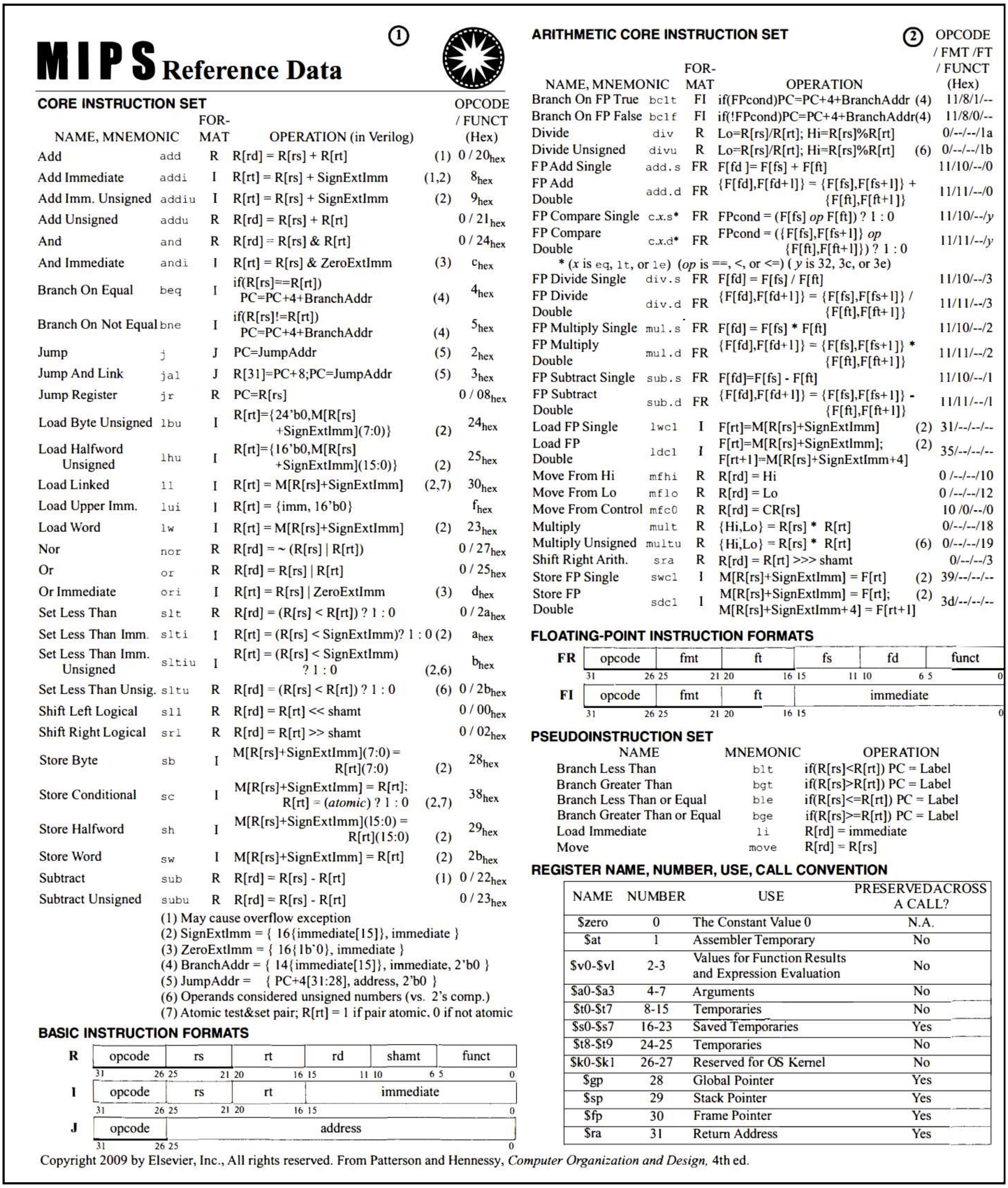
Appendix A: ASCII Table (Section 2.2)



ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Appendix B: MIPS Reference Page (Section 3.2)



MIPS Reference Data

①

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)

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
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		

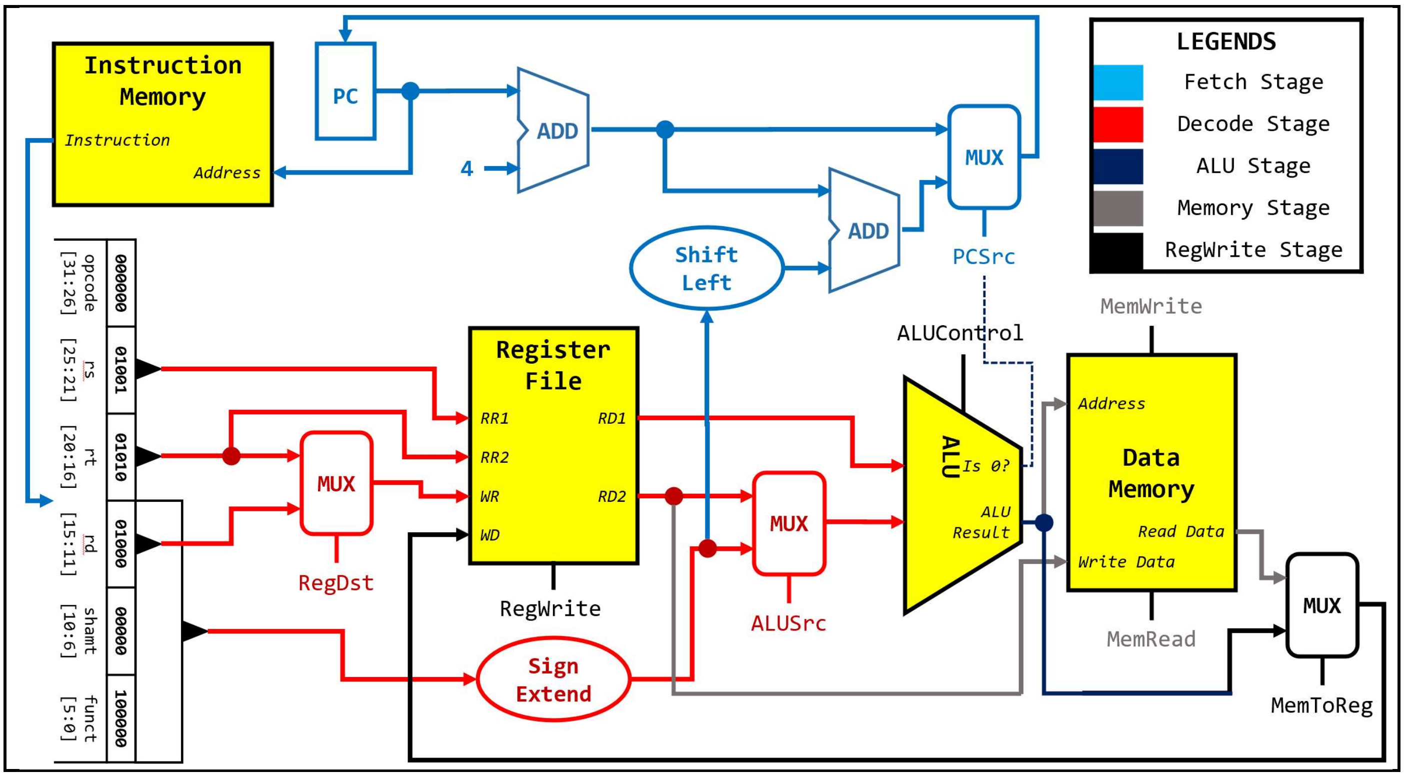
PSEUDOINSTRUCTION SET

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	PRESERVEDACROSS 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

Appendix C: Complete Datapath (Section 4.2)


Instruction Memory
Instruction
Address
PC
ADD
4
ADD
MUX
PCSrc
Shift Left
LEGENDS
🟦 Fetch Stage
🟥 Decode Stage
🟦 ALU Stage
⬜ Memory Stage
⬛ RegWrite Stage
000000 opcode [31:26]
01001 rs [25:21]
01010 rt [20:16]
01000 rd [15:11]
00000 shamt [10:6]
100000 funct [5:0]
Register File
RR1
RR2
WR
WD
RD1
RD2
MUX
RegDst
RegWrite
MUX
ALUSrc
Sign Extend
ALUControl
ALU
Is 0?
ALU Result
MemWrite
Data Memory
Address
Read Data
Write Data
MemRead
MUX
MemToReg