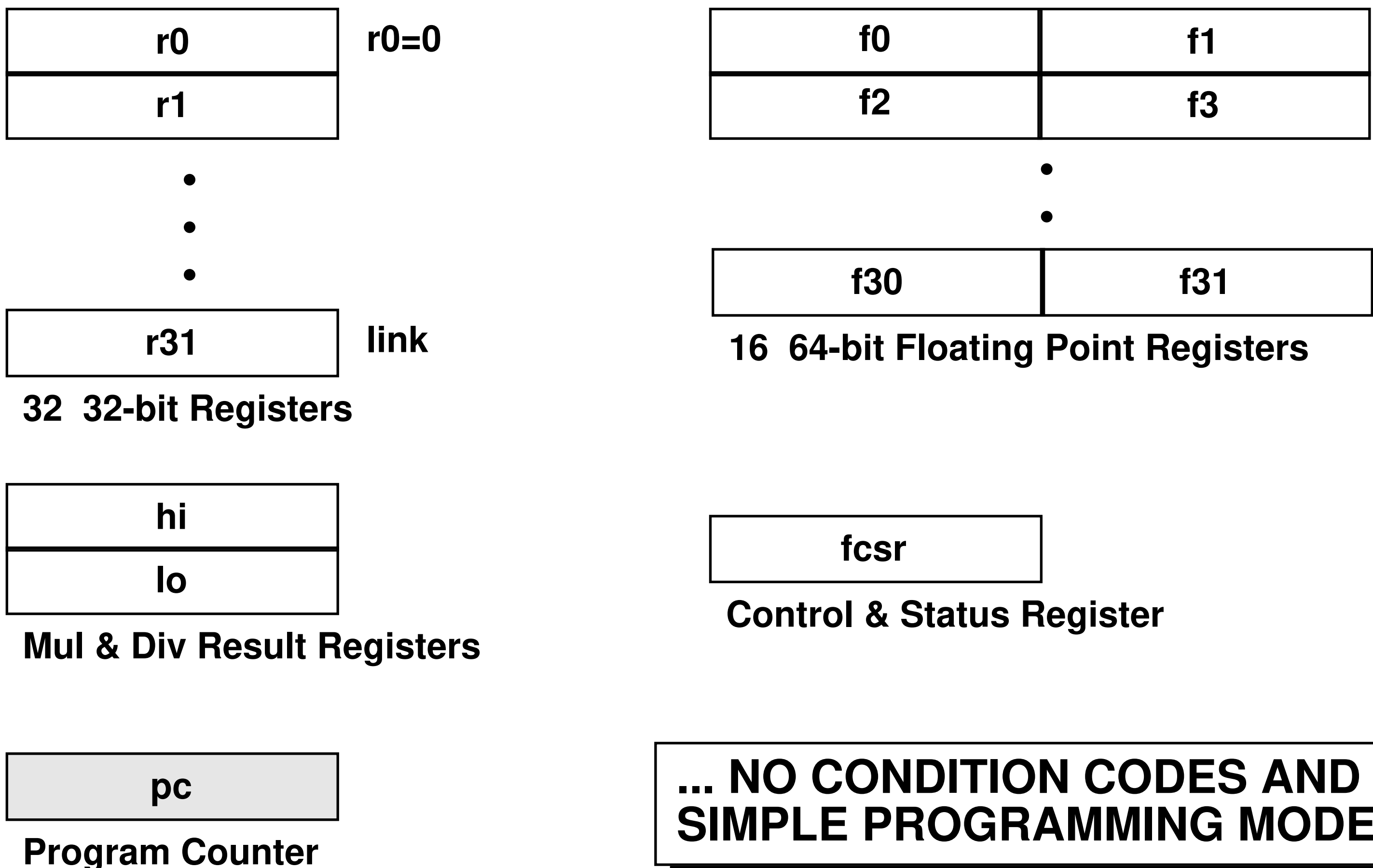
User State Registers



... NO CONDITION CODES AND A SIMPLE PROGRAMMING MODEL.

Instruction Formats

I-type: ALU immediate, Memory Op and Branch

op	rs	rt	immediate
6	5	5	16

R-type: Register to Register

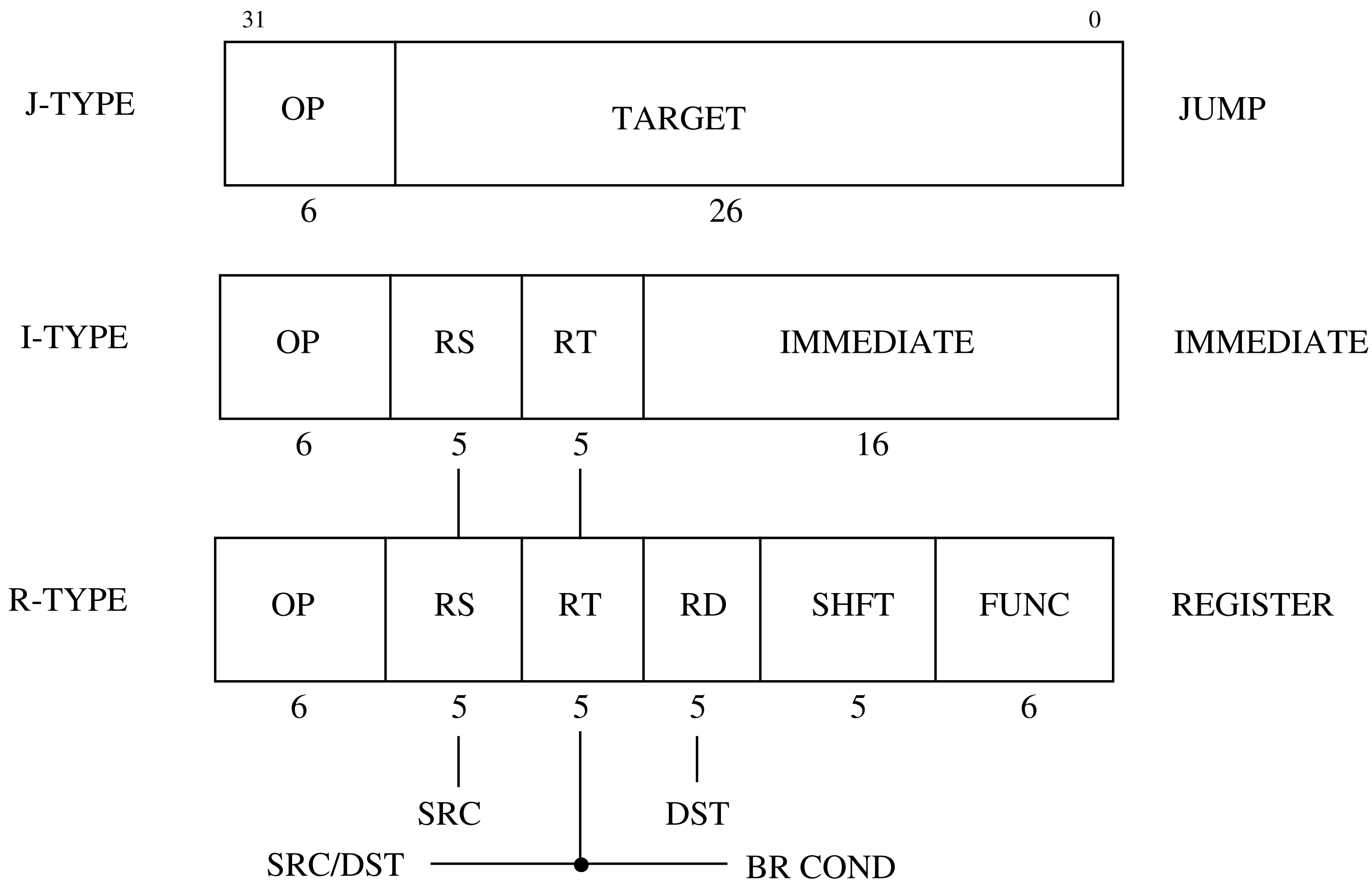
op	rs	rt	rd	samt	func
6	5	5	5	5	6

J-type: Long jump, word addressed

op	target
6	26

SIMPLE, REGULAR FORMATS ALLOW FAST DECODE & PIPELINE

LR3000 INSTRUCTION FORMATS



- All Instructions are 32 Bits

INSTRUCTION SET

LOAD/STORE

Byte addressed, Bi-endian

Word, Halfword, Byte

Signed, Unsigned

Unaligned References

Base + 16 bit Offset

ALU

Add, Sub, Logicals

- Rd:=Rs op Rt, 3 register operations
- Rt:=Rs op I, 2 register + 16 bit immediate

Impact of compare & branch on arithmatic ops:

- Add/Sub with no trap, unsigned arith
- Add/Sub with trap, ADA, Pascal, LISP

BRANCHES

No Condition Codes; Compare and Branch

One Instruction Branch on:

- A<0, A 0, A>0, A 0, A=B, A B

When needed, two instructions for:

- A<B, A B, A>B, A B

Branches execute next instruction before branching

Also Jumps: J, JAL, JR, JALR

MULTIPLY/DIVIDE

Compile most constants with shft/add/sub

Hardware accelerates remaining

- 12 cycle mult, 35 cycle divide

64 bit product, or quotient/remainder

ARITHMETIC INSTRUCTIONS

	SIGNED		UNSIGNED	
	Reg	Imm	Reg	Imm
ADD	ADD	ADDI	ADDU	ADDIU
SUBTRACT	SUB	X	SUBU	X
MULTIPLY	MULT	X	MULTU	X
DIVIDE	DIV	X	DIVU	X
SHIFT RIGHT	X	X	SRAV	SRA

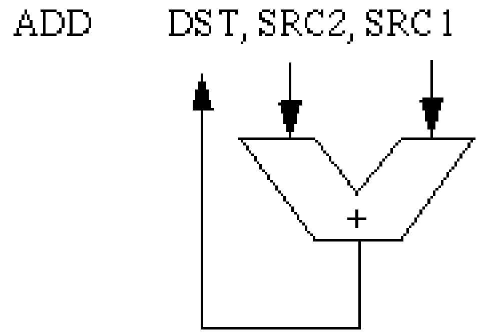
- Immediate Value is 16-Bits, Sign Extended
- Signed Instructions Generate Exception on Overflow

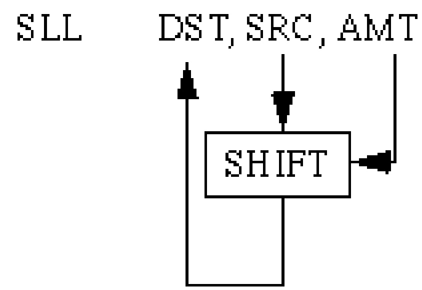
LOGICAL INSTRUCTIONS

	Reg	Imm
AND	AND	ANDI
OR	OR	ORI
NOR	NOR	X
EXCL-OR	XOR	XORI
SHIFT LEFT	SLLV	SLL
SHIFT RIGHT	SRLV	SRL

- Immediate Value is 16-Bits, Zero Extended

THREE OPERAND INSTRUCTIONS





- ADD DST, SRC when DST = SRC2

LOAD CONSTANT

Load 0x12345678 into R8 **(LI $8, 0x12345678)**

Zero Fill

LUI $8, 0x1234	$8	1 2 3 4 | 0 0 0 0
ORI $8, 0x5678	$8	1 2 3 4 | 5 6 7 8

Load 0x1234 into R8 **(LI $8, 0x1234)**

Sign Extended

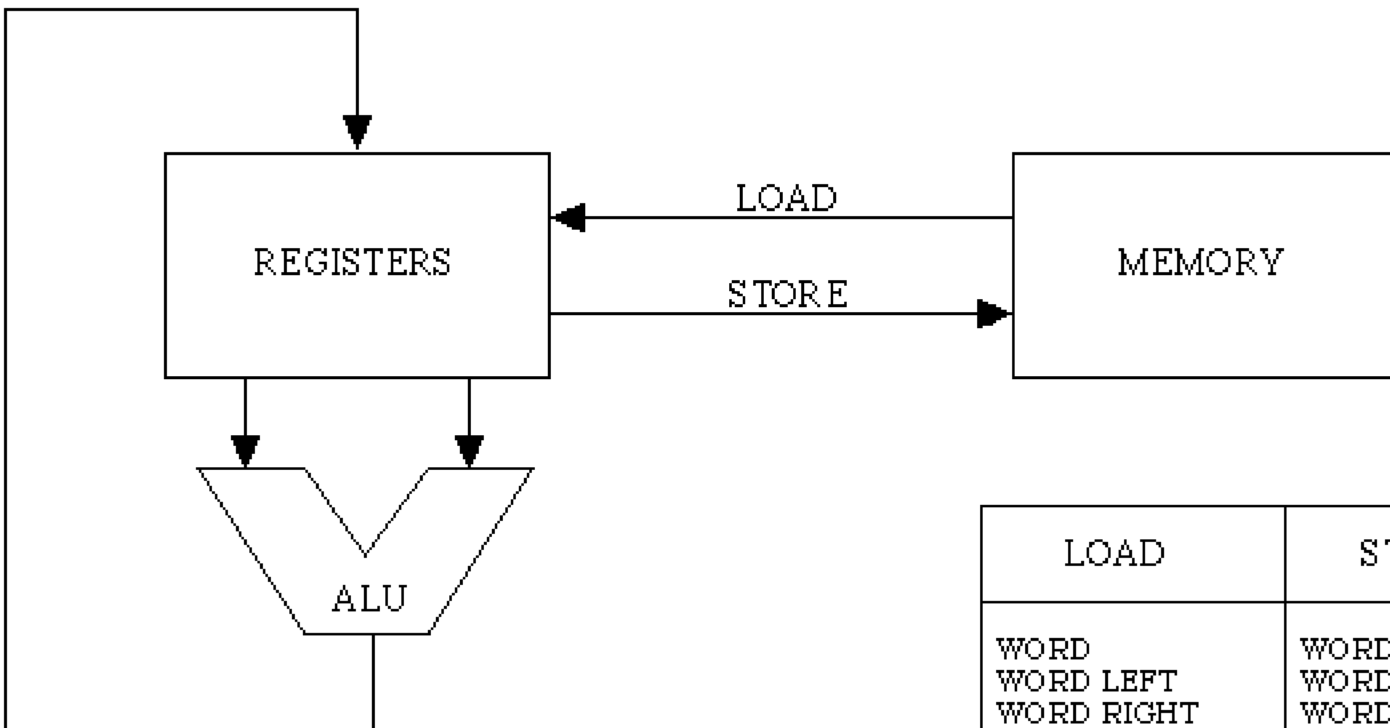
ADDIU $8, $0, 0x1234	$8	0 0 0 0 | 1 2 3 4

Load -1 into R8 **(LI $8, -1)**

Sign Extended

ADDIU $8, $0, -1	$8	F F F F | F F F F

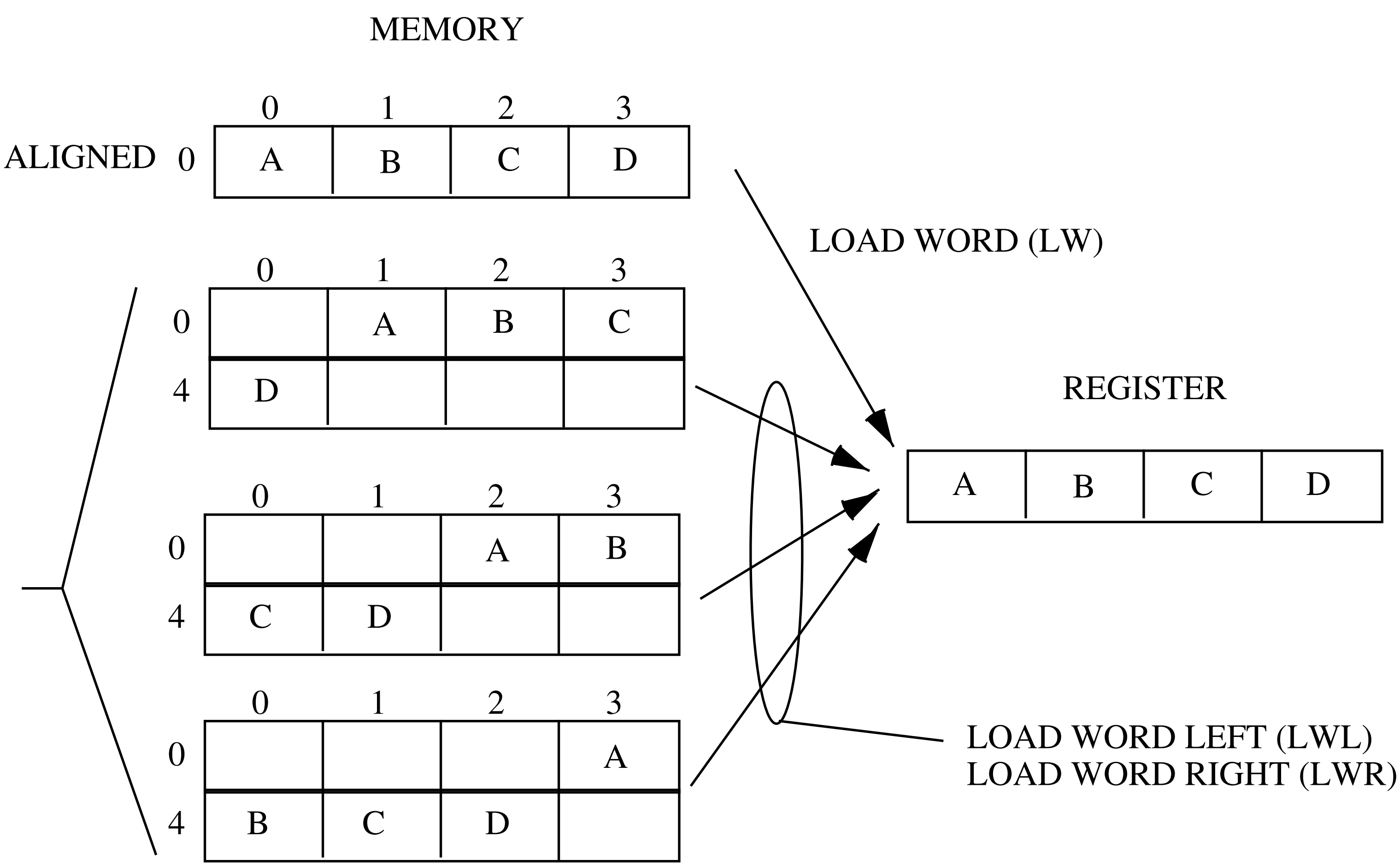
LOAD/STORE ARCHITECTURE



LOAD	STORE
WORD WORD LEFT WORD RIGHT H/W + UNSIGNED BYTE + UNSIGNED	WORD WORD LEFT WORD RIGHT HALF WORD BYTE

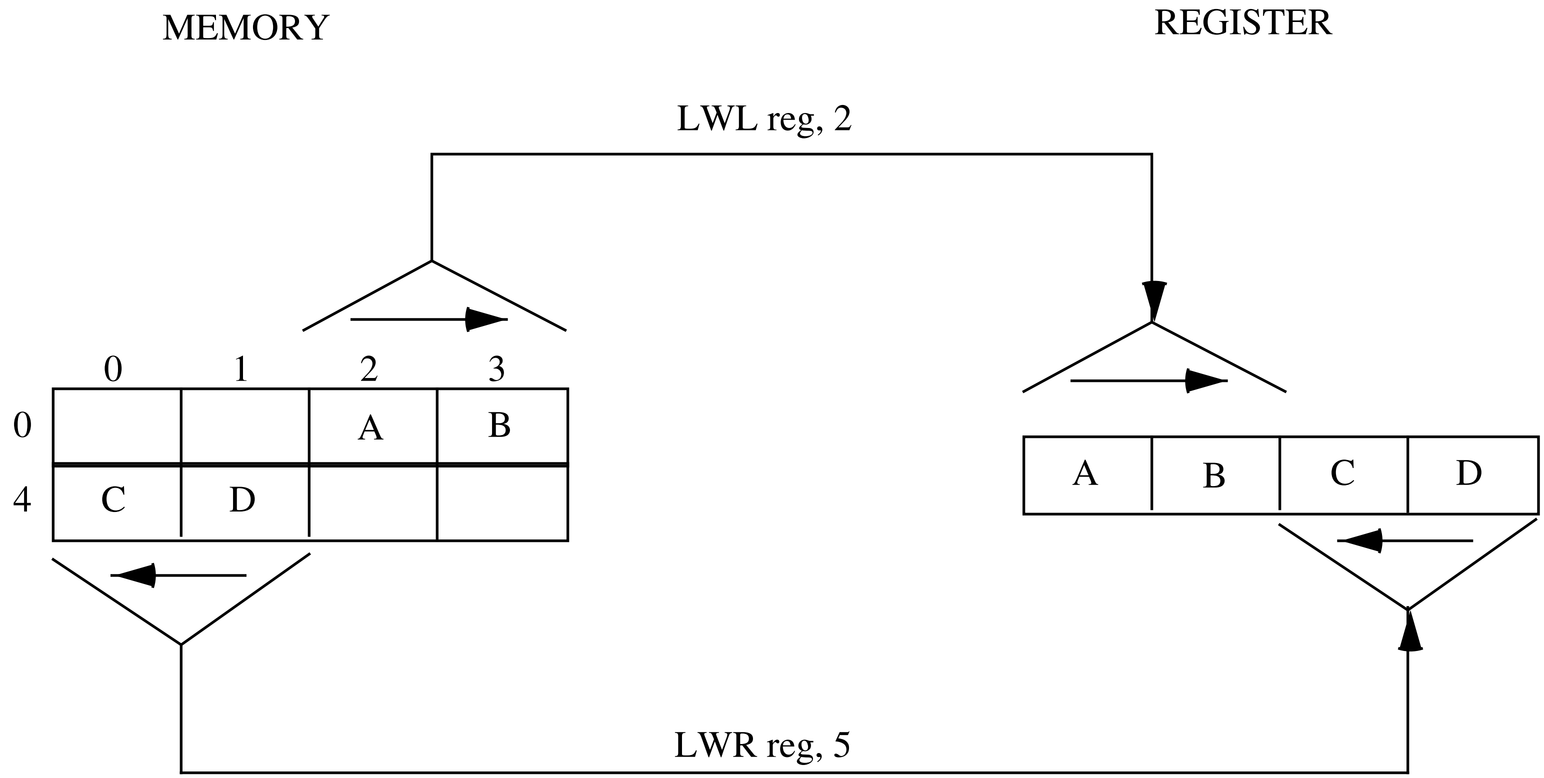
- Load Always Gets a Word on the Bus but Ignores Bytes it Doesn't Need

LOAD WORD



- Assumes Big-Endian

LOAD WORD UNALIGNED



- LWL/LWR - Load from Byte Address to Word Boundary
- Assumes Big-Endian

BYTE ORDER MODE SELECTION

Big Endian Ordering: the address of a word or half-word corresponds to its most significant byte (MSB). Used by M68000 and IBM mainframes.

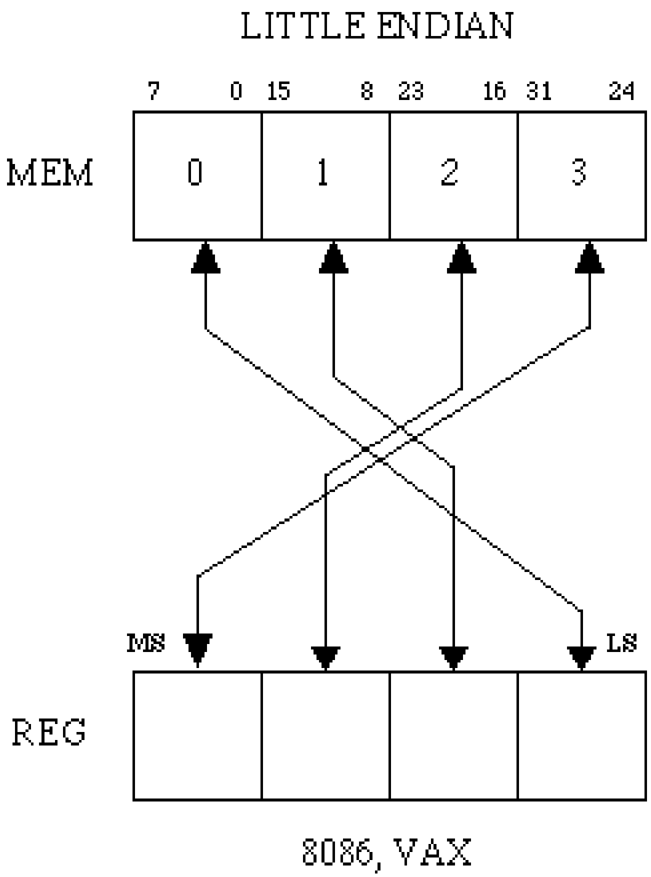
	31 24	23 16	15 8	7 0
label:	0	1	2	3

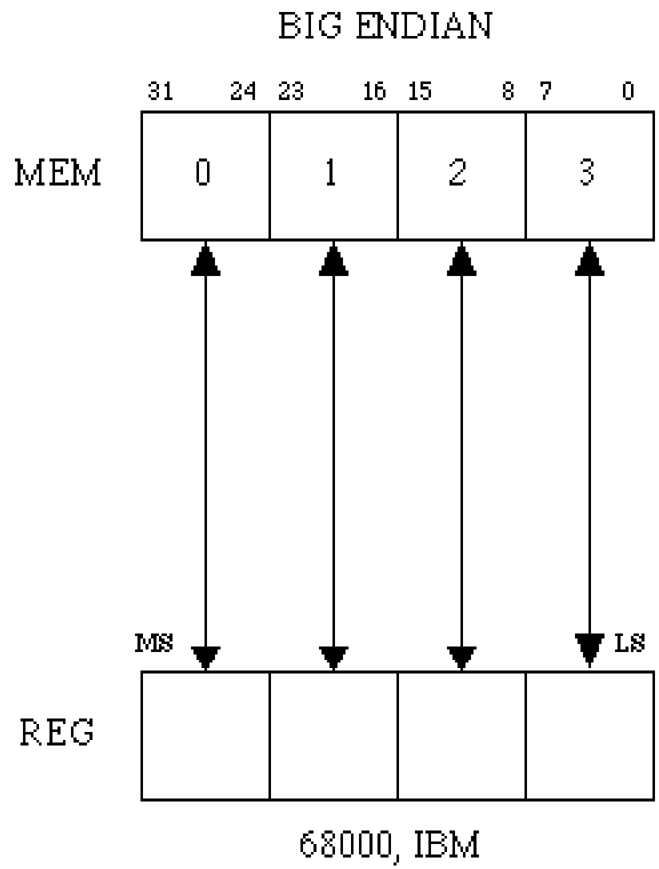
Little Endian Ordering: the address of a word or half-word corresponds to its least significant byte (LSB). Used by i8086 and VAX.

	7 0	15 8	23 16	31 24
label:	0	1	2	3

The names Big Endian and Little Endian are used because of the apt analogy to the two feuding islands in Gulliver's Travels by Jonathan Swift, and were coined by Danny Cohen in his article, "On Holy Wars and a Plea for Peace, " published in Computer, Vol. 14, No. 10, Oct 1981, pp. 49-54.

BYTE ORDER MODE SELECTION





- Only Affects Word, Tribyte & Half-Word Operations

ADDRESS OF BYTES WITHIN WORDS
(FROM LR3000 DATA SHEET)

BIG ENDIAN

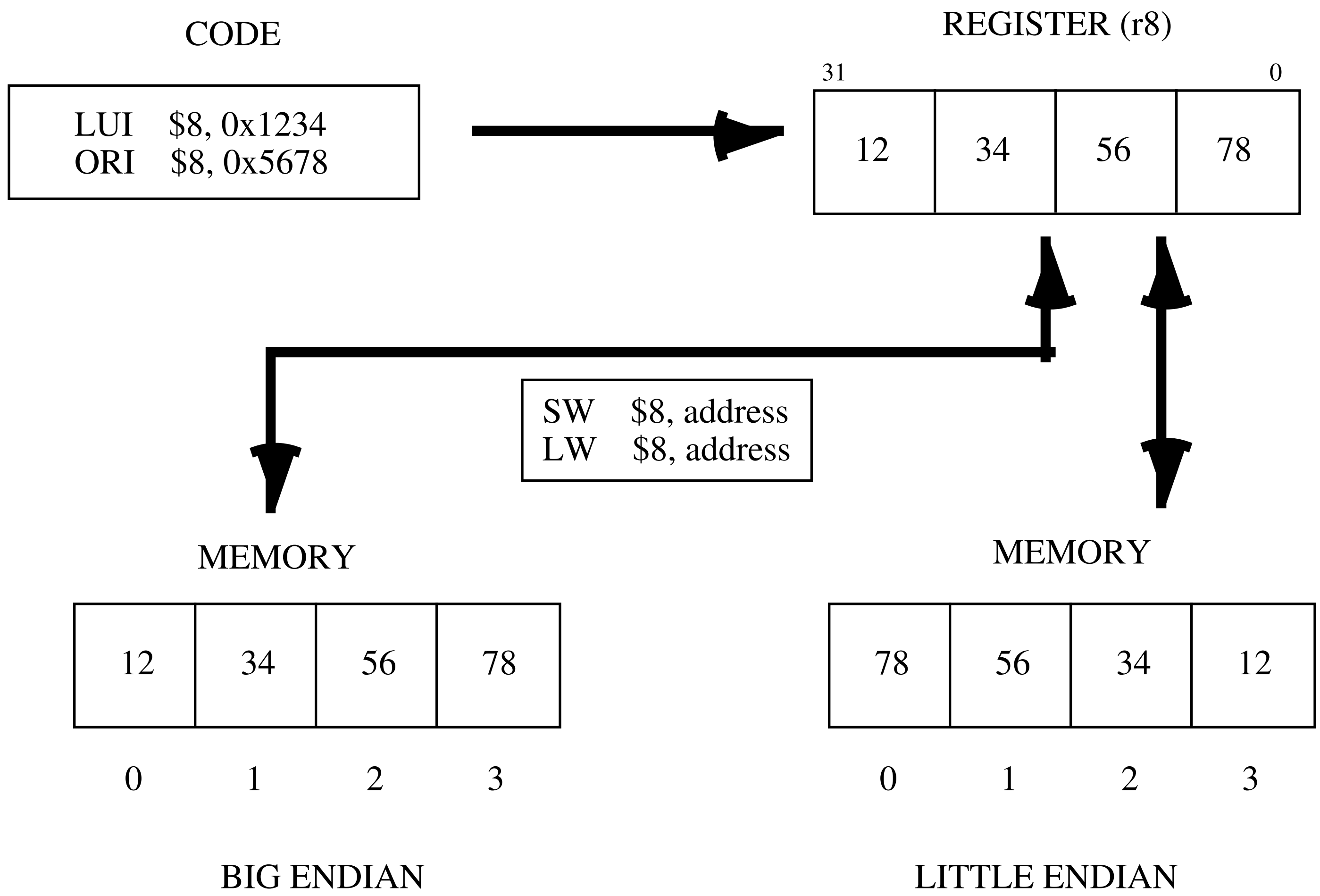
	31 24	23 16	15 8	7 0	Word Address
Higher Address	8	9	10	11	8
	4	5	6	7	4
Lower Address	0	1	2	3	0

LITTLE ENDIAN

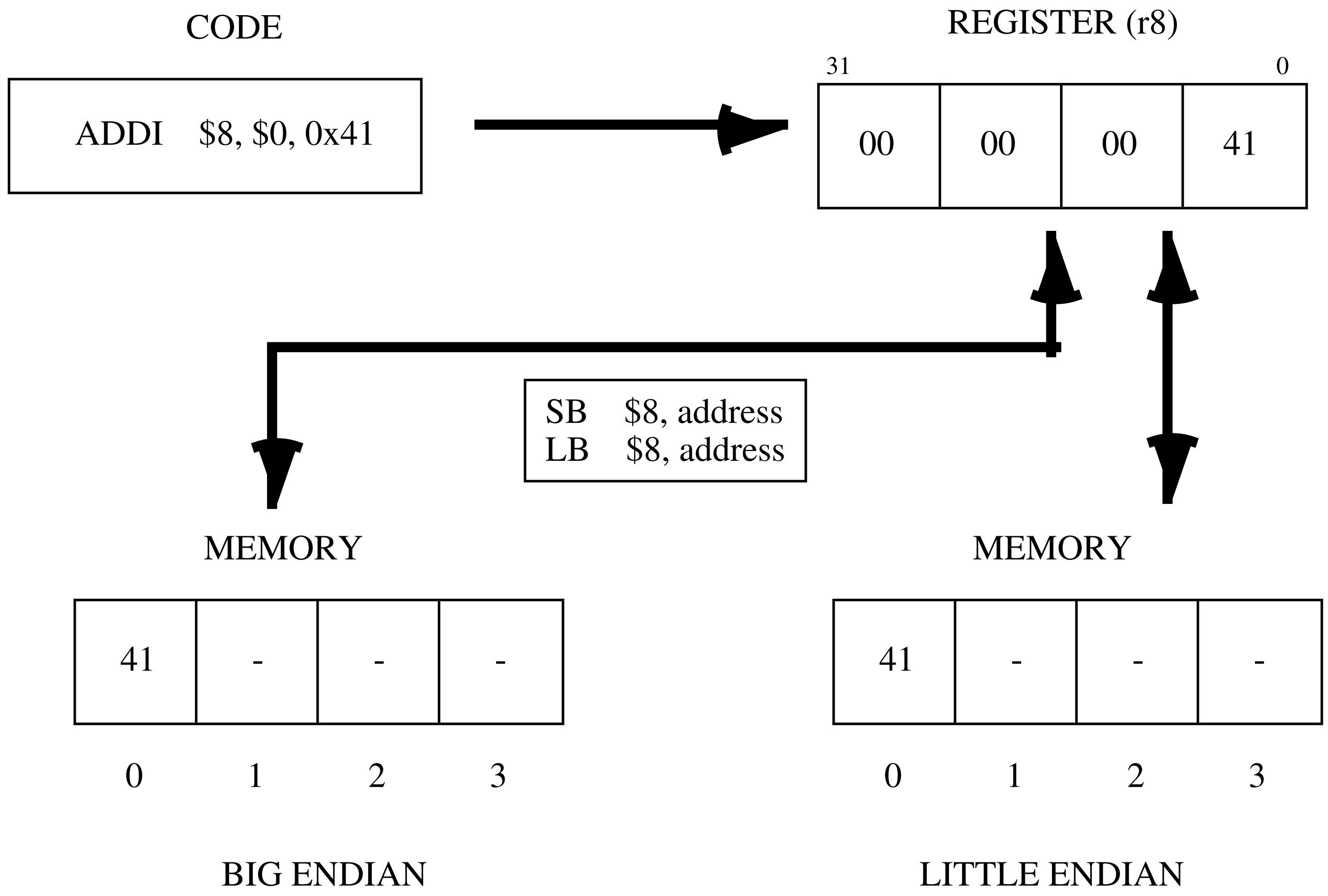
	31 24	23 16	15 8	7 0	Word Address
Higher Address	11	10	9	8	8
	7	6	5	4	4
Lower Address	3	2	1	0	0

- Same Information, Different Form

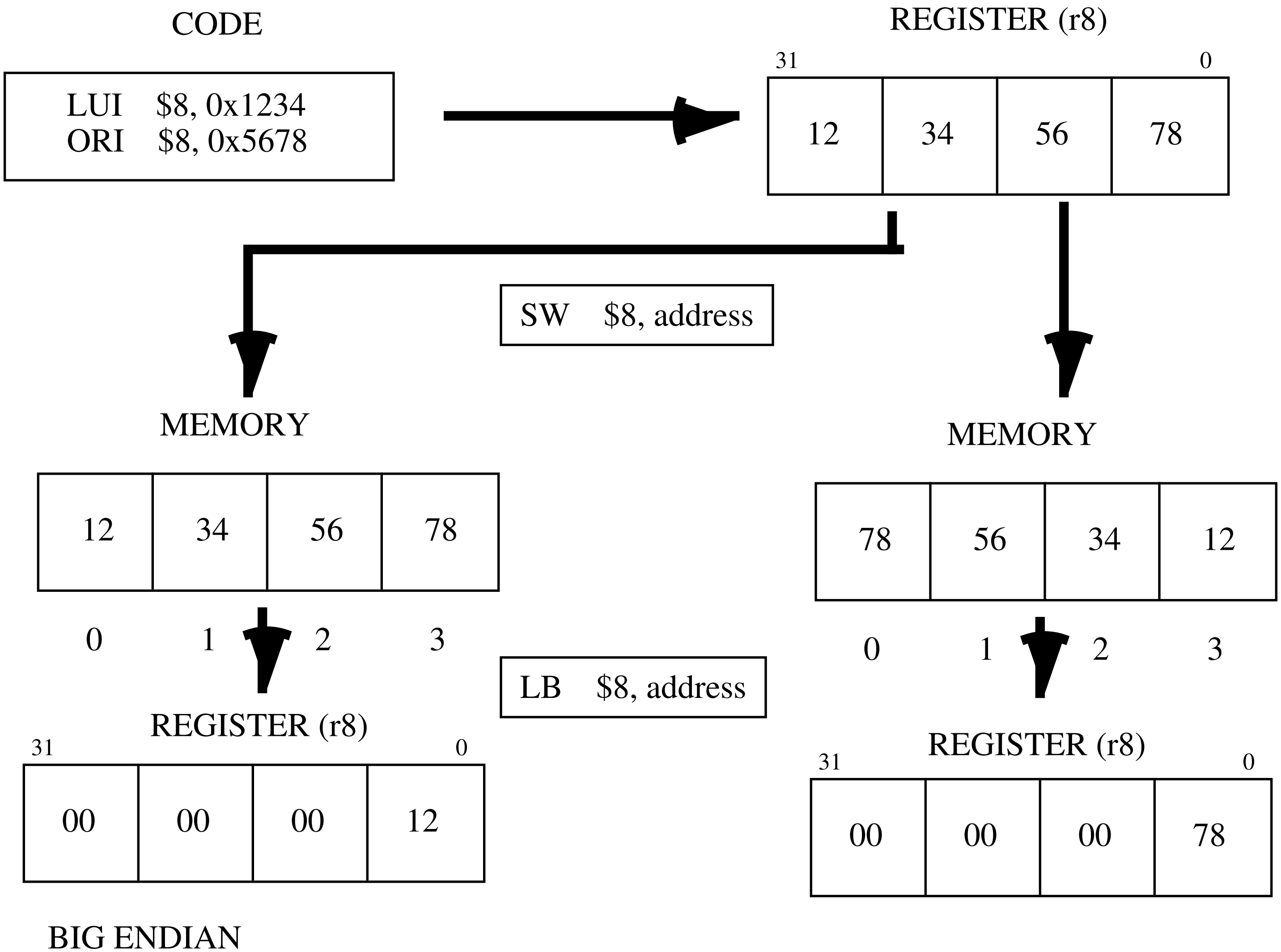
LOAD/STORE WORD



LOAD/STORE BYTE



STORE WORD - LOAD BYTE



BIG & LITTLE ENDIAN DATA

CODE

```
int  i = 0x12345678;

struct  {
            short j;
            short k;
          }  sht = {0x1234, 0x5678};

char  str[] = "ABC";
```

BIG ENDIAN MEMORY

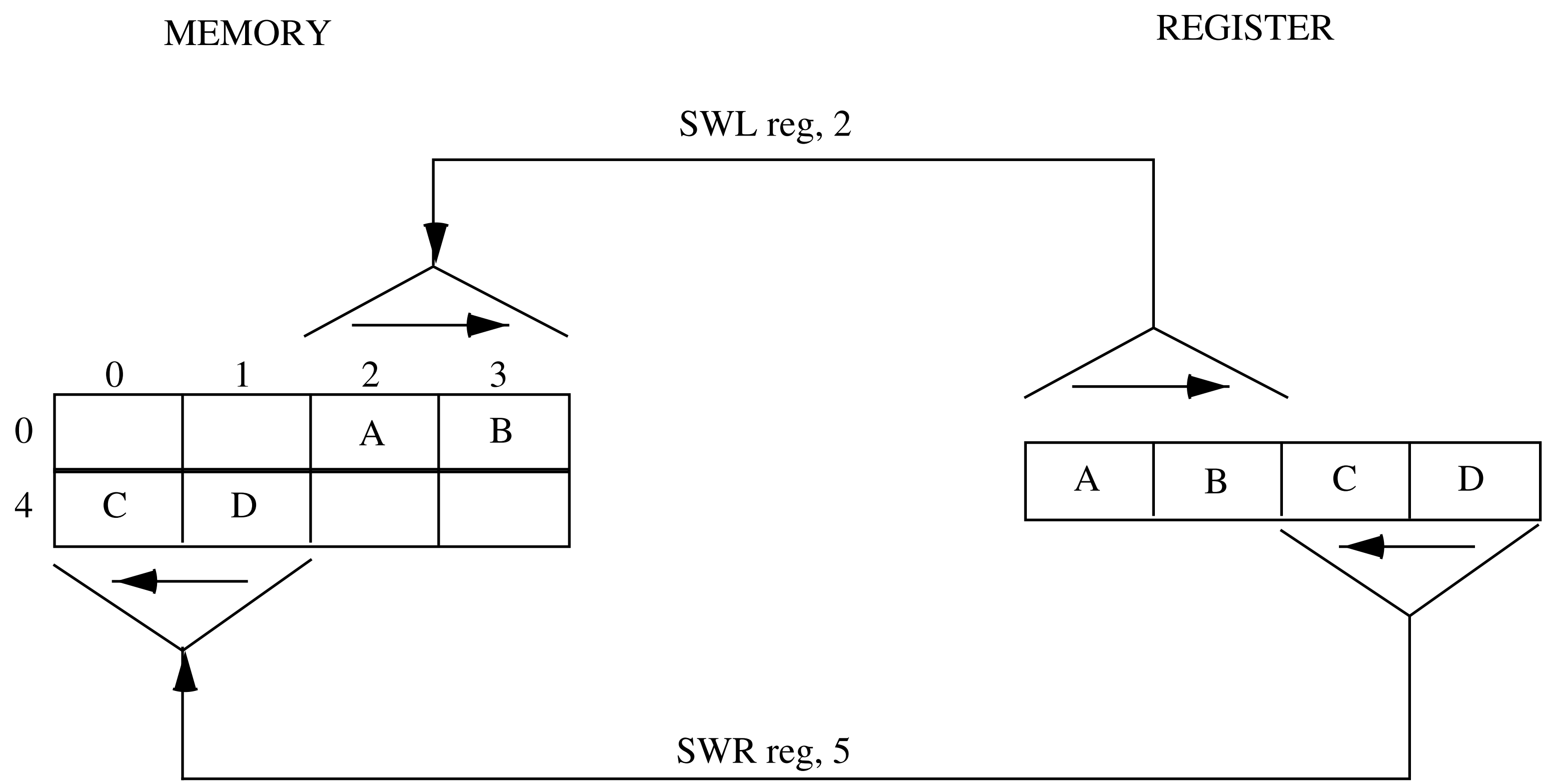
	0	1	2	3
0	12	34	56	78
4	12	34	56	78
8	41	42	43	00

LITTLE ENDIAN MEMORY

	0	1	2	3
0	78	56	34	12
4	34	12	78	56
8	41	42	43	00

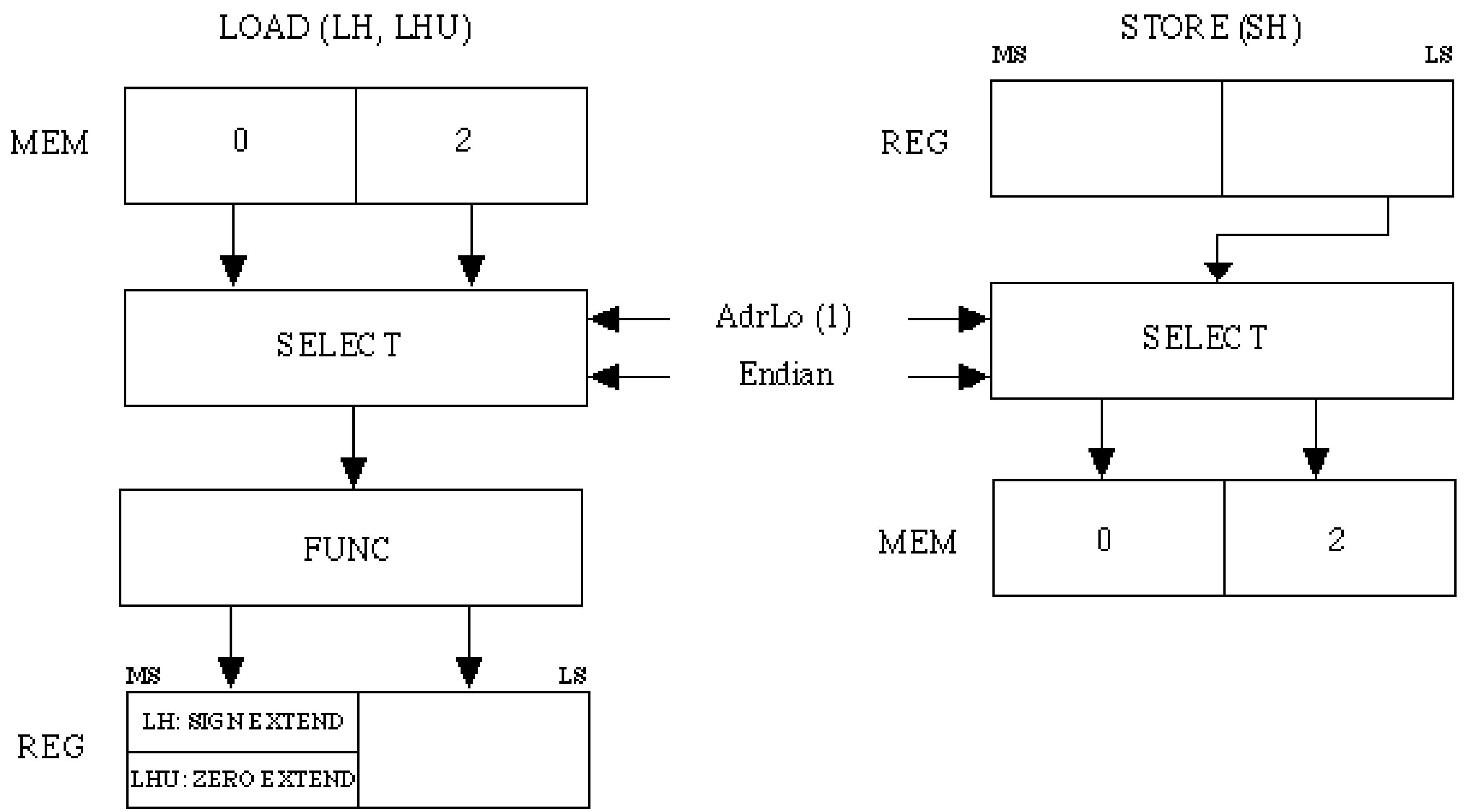
- Interpretation of Data in Memory Depends on Data type.

STORE WORD UNALIGNED

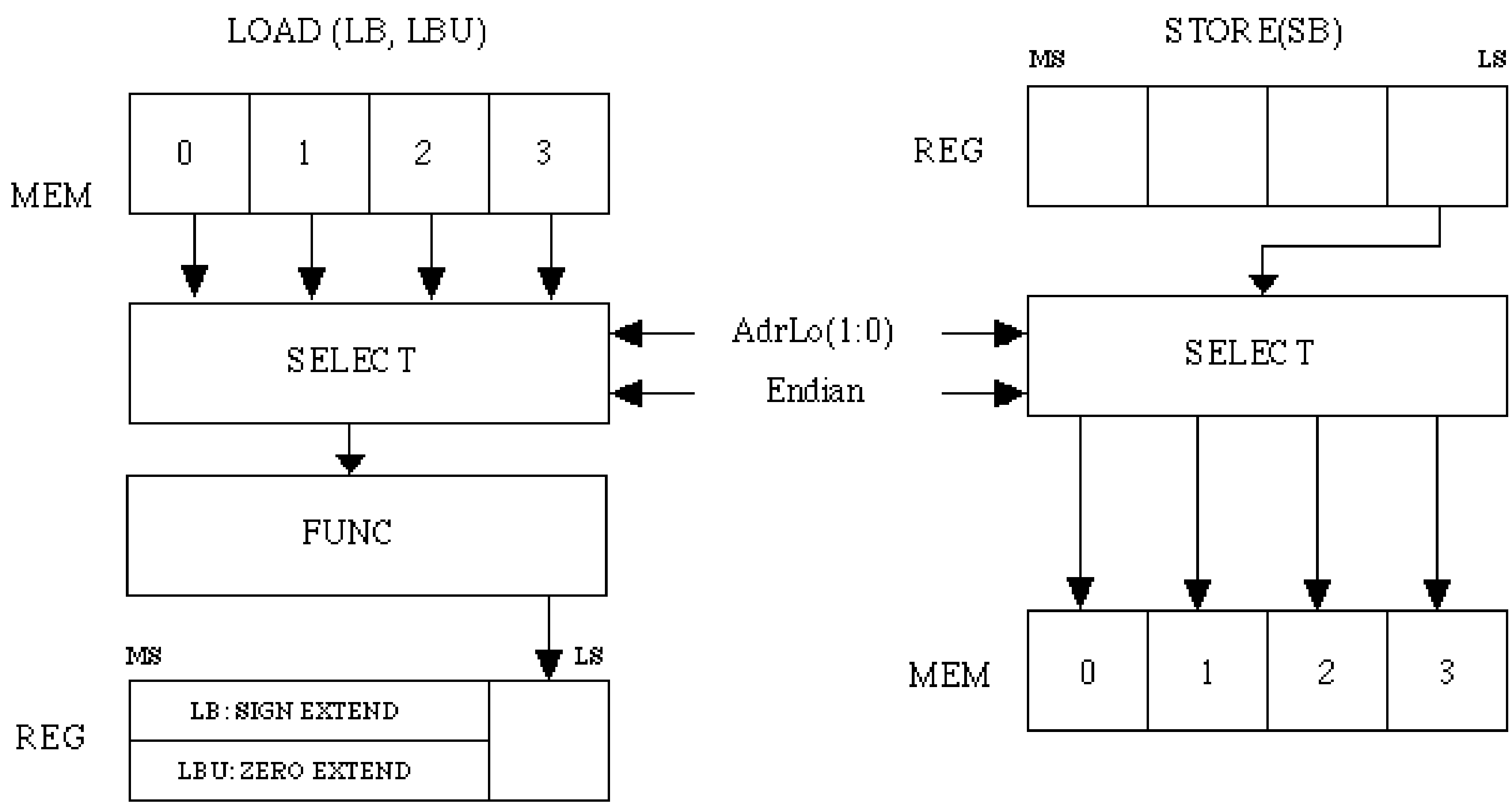


- SWL/SWR - Store from Byte Address to Word Boundary
- Assumes Big-Endian

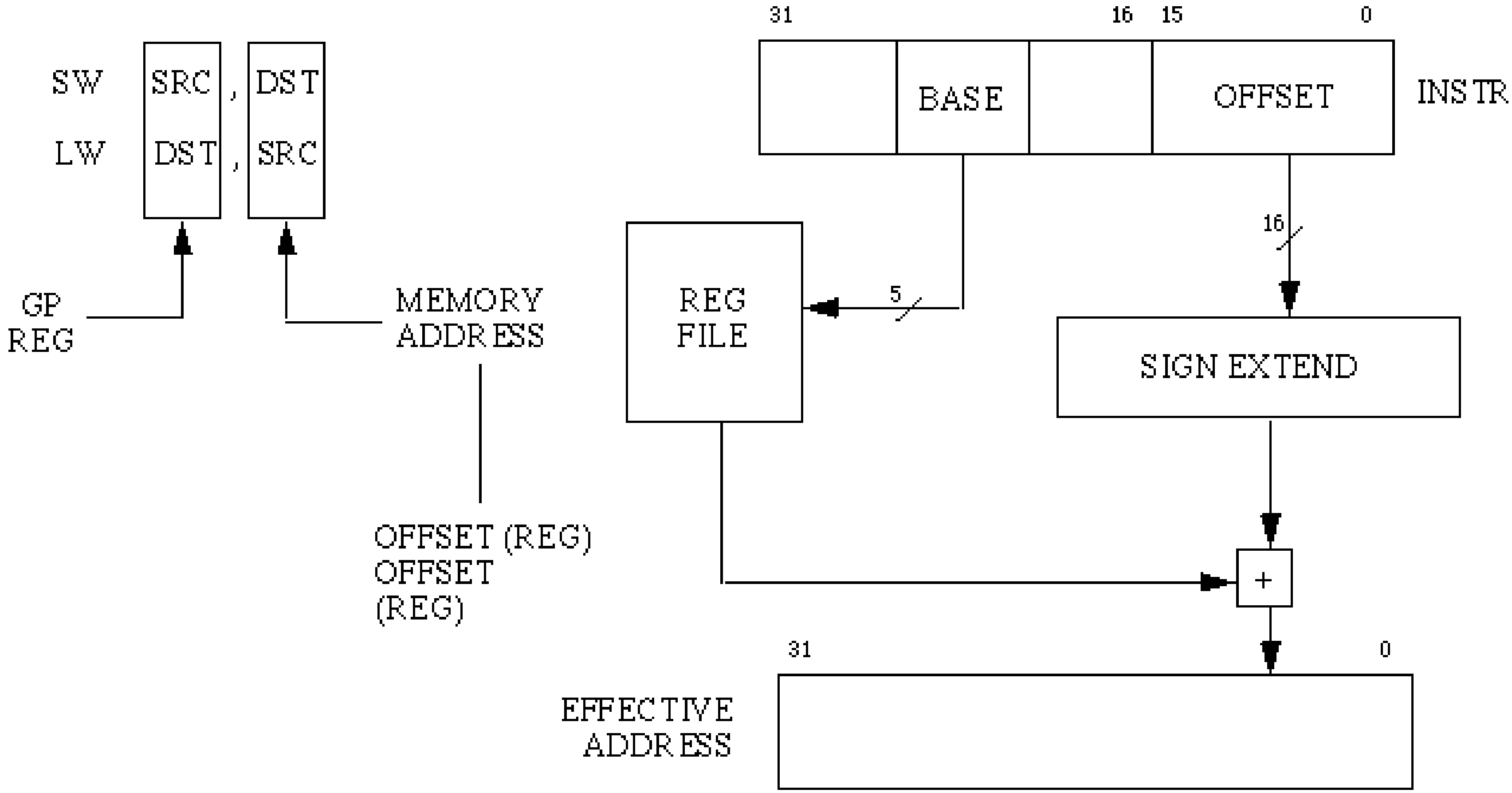
LOAD/STORE HALF-WORD



LOAD/STORE BYTE

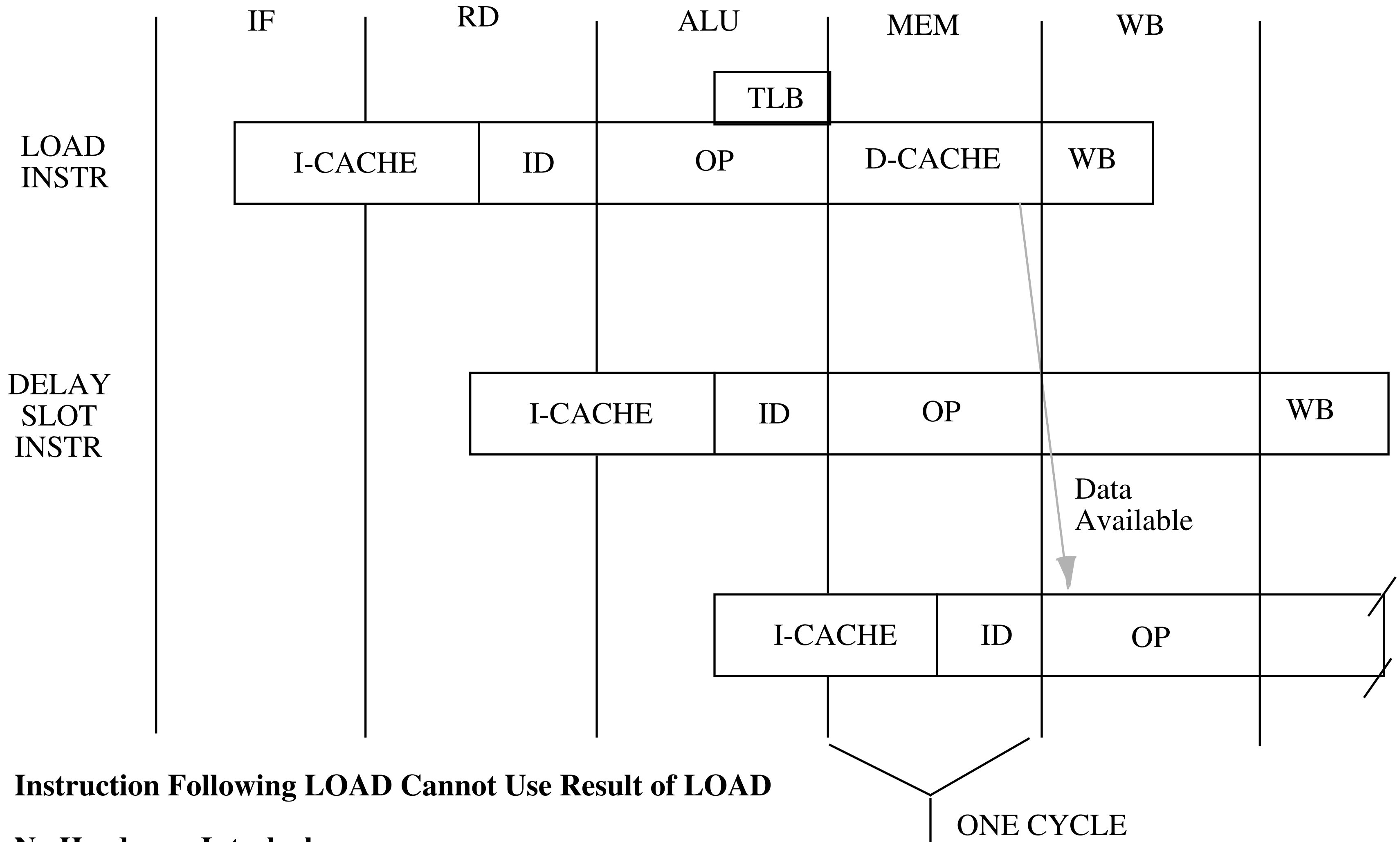


LOAD/STORE ADDRESSING MODE



- Range : ± 32KB from Base
- $gp (R28) Points to Program's Data Space

LOAD INSTRUCTION DELAY SLOT



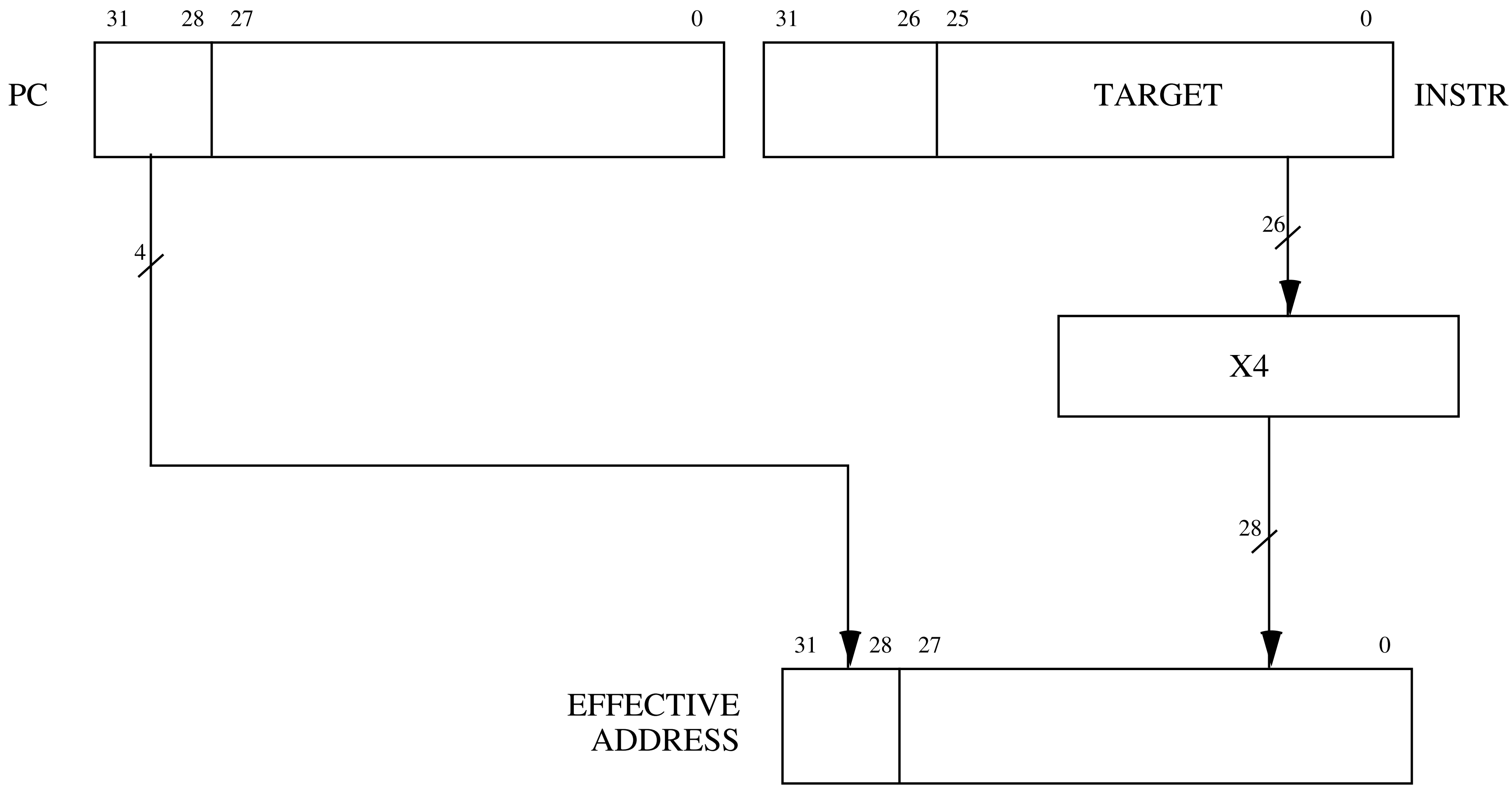
- **Instruction Following LOAD Cannot Use Result of LOAD**
- **No Hardware Interlock**

CONTROL TRANSFER INSTRUCTIONS

	UNCOND		COND
	Reg	Imm	
JUMP	JR	J	Bcc
CALL	JALR	JAL	BccAL

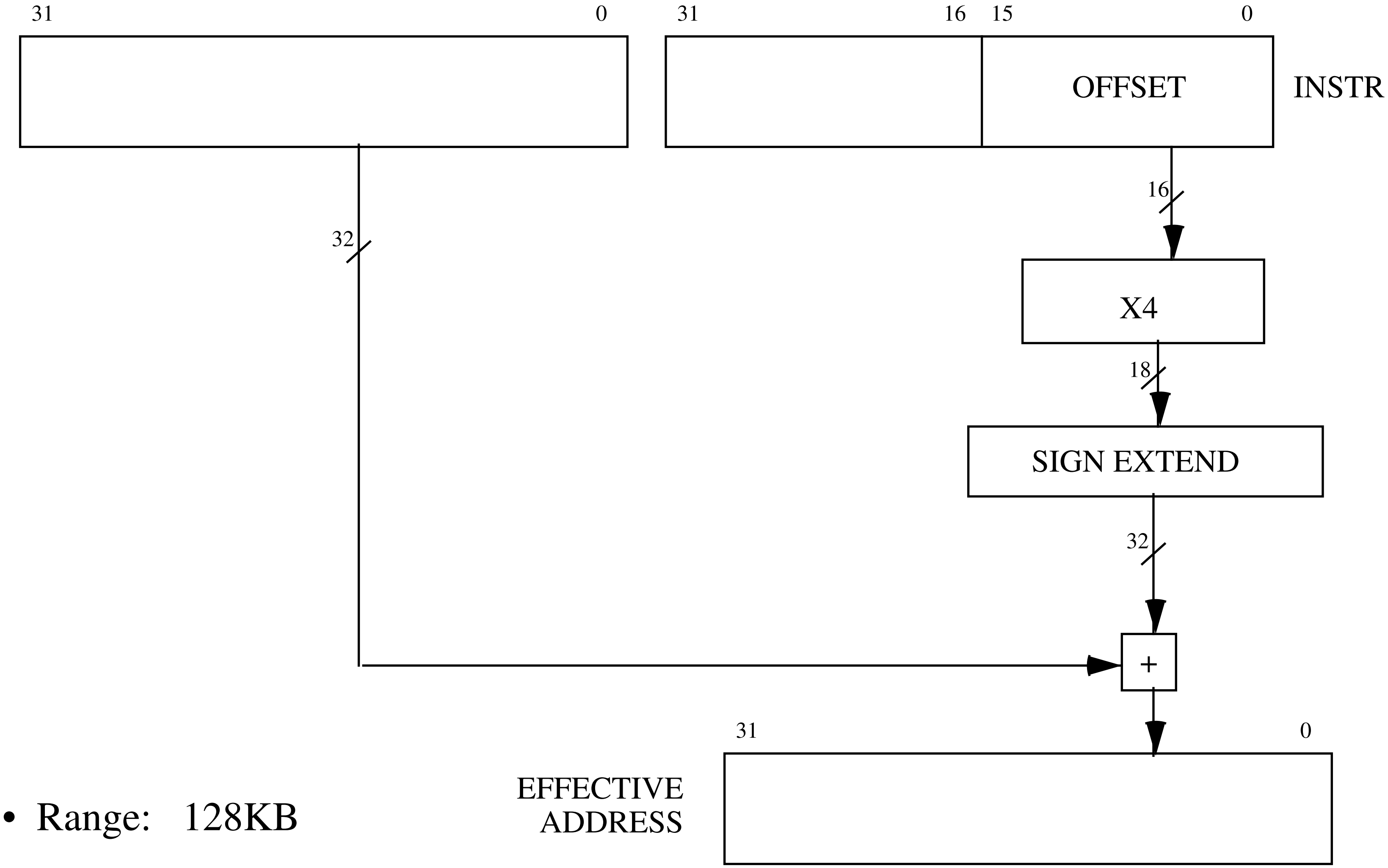
- Jumps are Unconditional
- Branches are Conditional
- AL = and Link

JUMP ADDRESSING MODE



- Jump within 256MB Page

BRANCH ADDRESSING MODE



- Range: 128KB

BRANCH INSTRUCTIONS

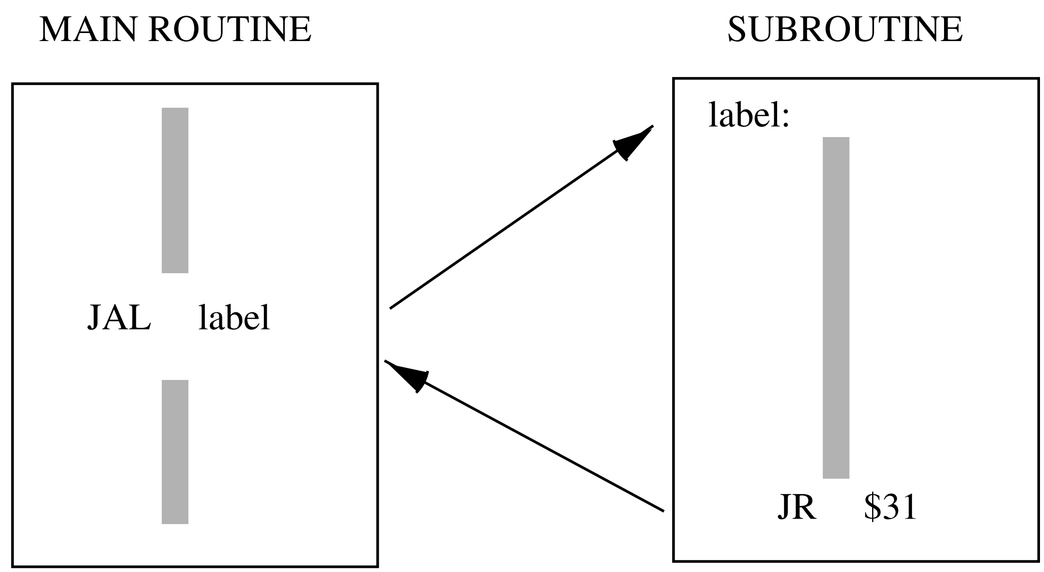
BEQ, BNE	SRC1, SRC2, LABEL	• Compare 2 Registers
BLTZ, BLEZ, BGEZ, BGTZ	SRC, LABEL	• Compare Register to Zero • Tests Sign Bit of SRC
BCzF, BCzT	LABEL	• Test CpCond(z) Input • z = 0 thru 3

COMPARE AND BRANCH

```
BGE    $8, $9, LABEL
          SLT   $1,  $8,  $9          or 16-Imm
          BEQ   $1,  $0,  LABEL
                                      Boolean TRUE or FALSE
```

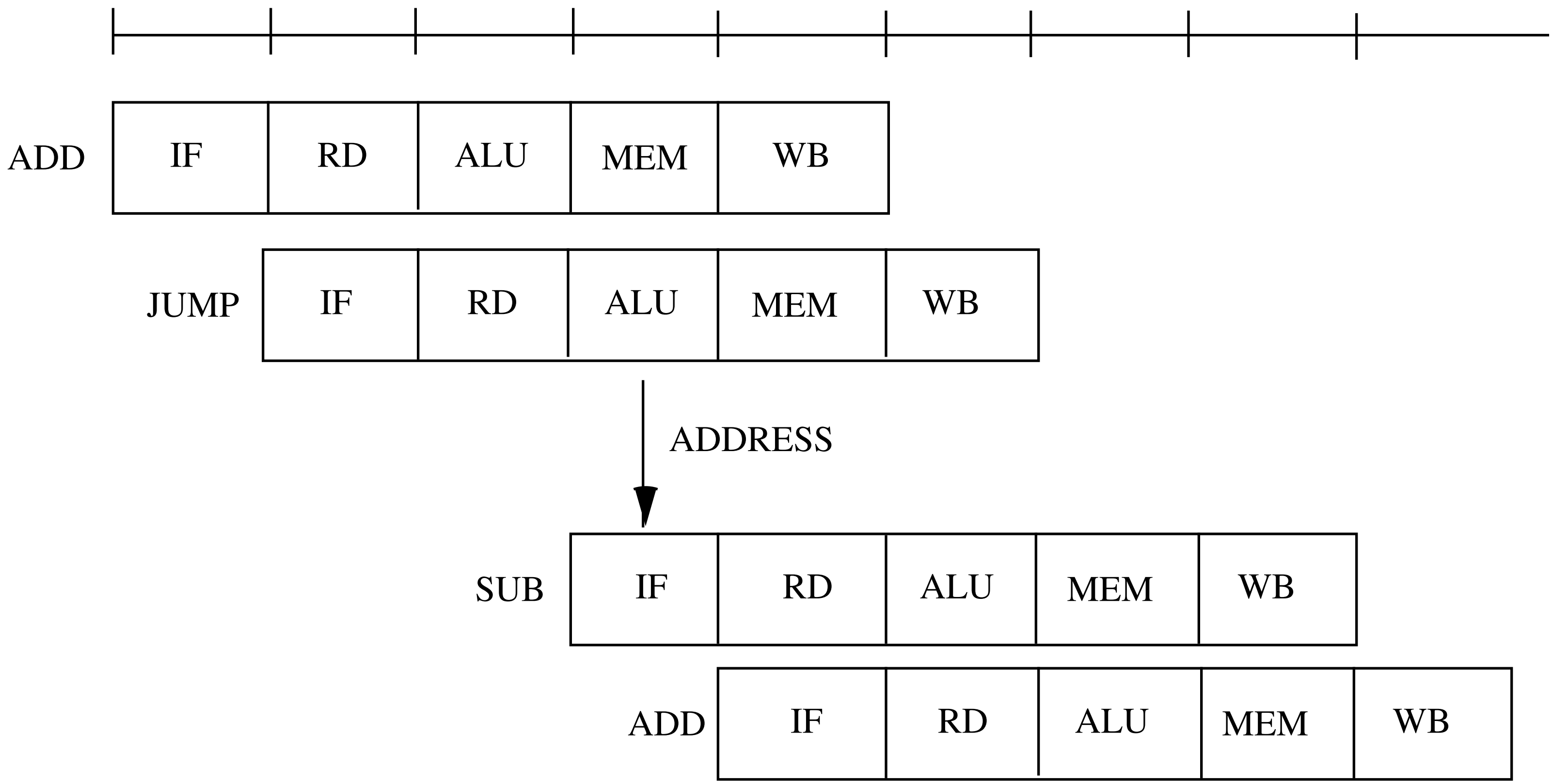
- SLT + (BEQ or BNE) Permits all Relations to be Tested
- $1 is Temp Register Reserved for Assembler
- $0 Contains Zero

SUBROUTINE CALLS



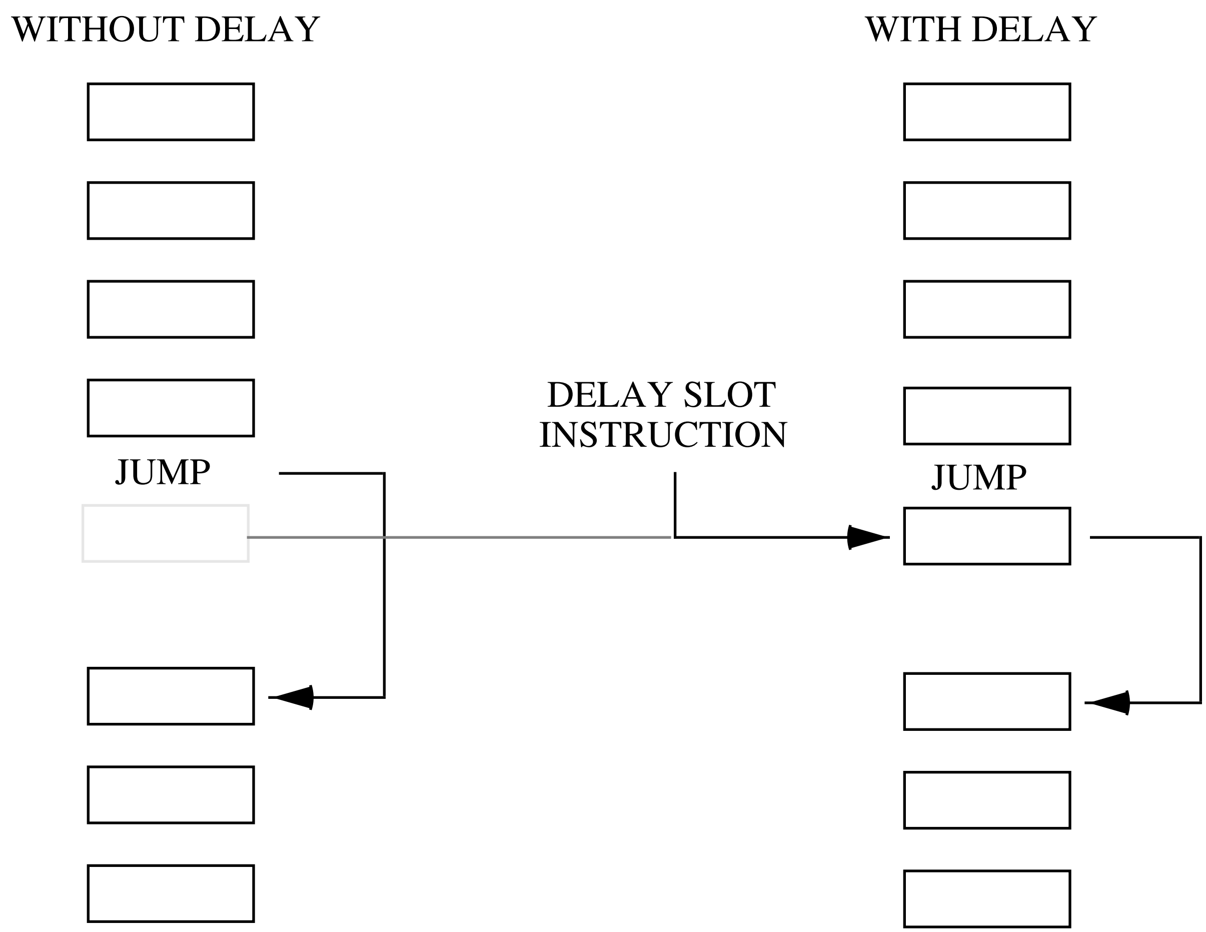
- JAL Stores Return Address in R31
- R31 Must be Saved Before Nested Call
- Return Uses Jump Indirect on R31

BRANCH WITHOUT A DELAY-SLOT INSTRUCTION



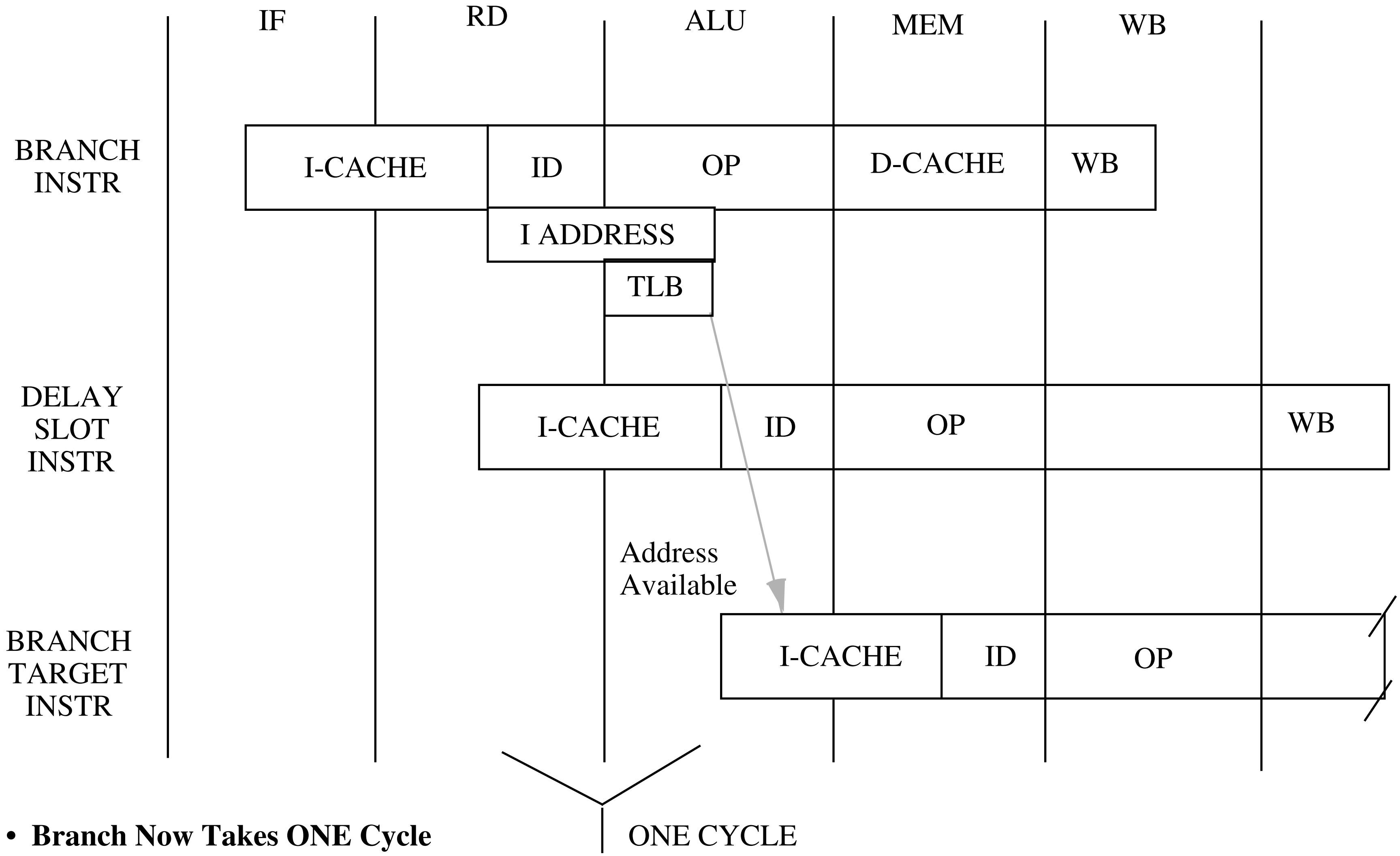
- Branch Would Take 2 Cycles

WHAT IS A DELAYED BRANCH ?



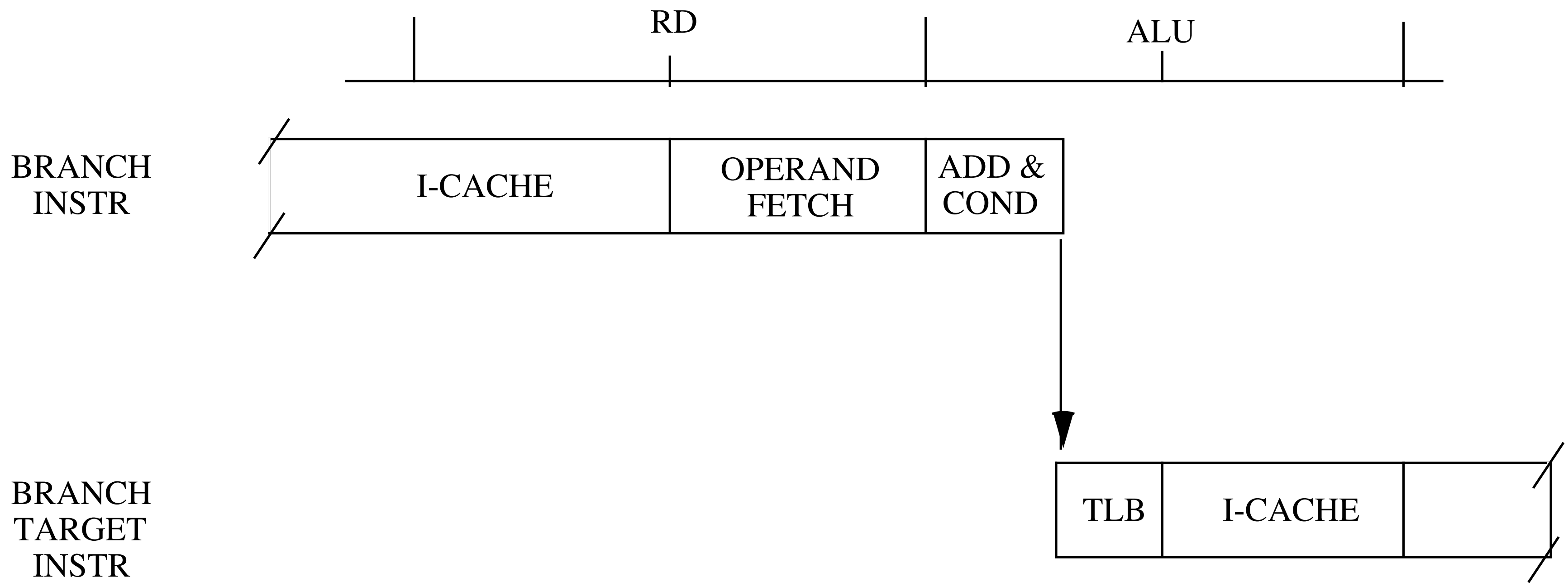
- Delay Slot Instruction is Always Executed

THE JUMP/BRANCH INSTRUCTION DELAY SLOT



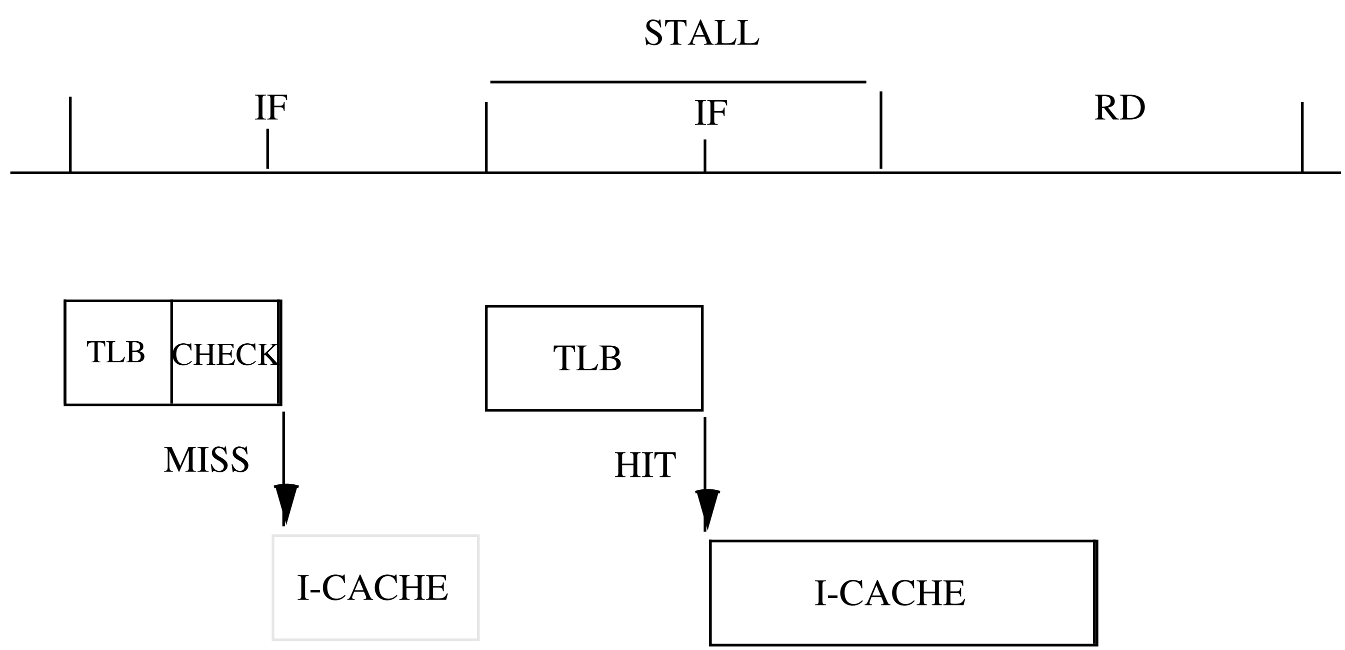
- **Branch Now Takes ONE Cycle**

DETAIL OF BRANCH OPERATION



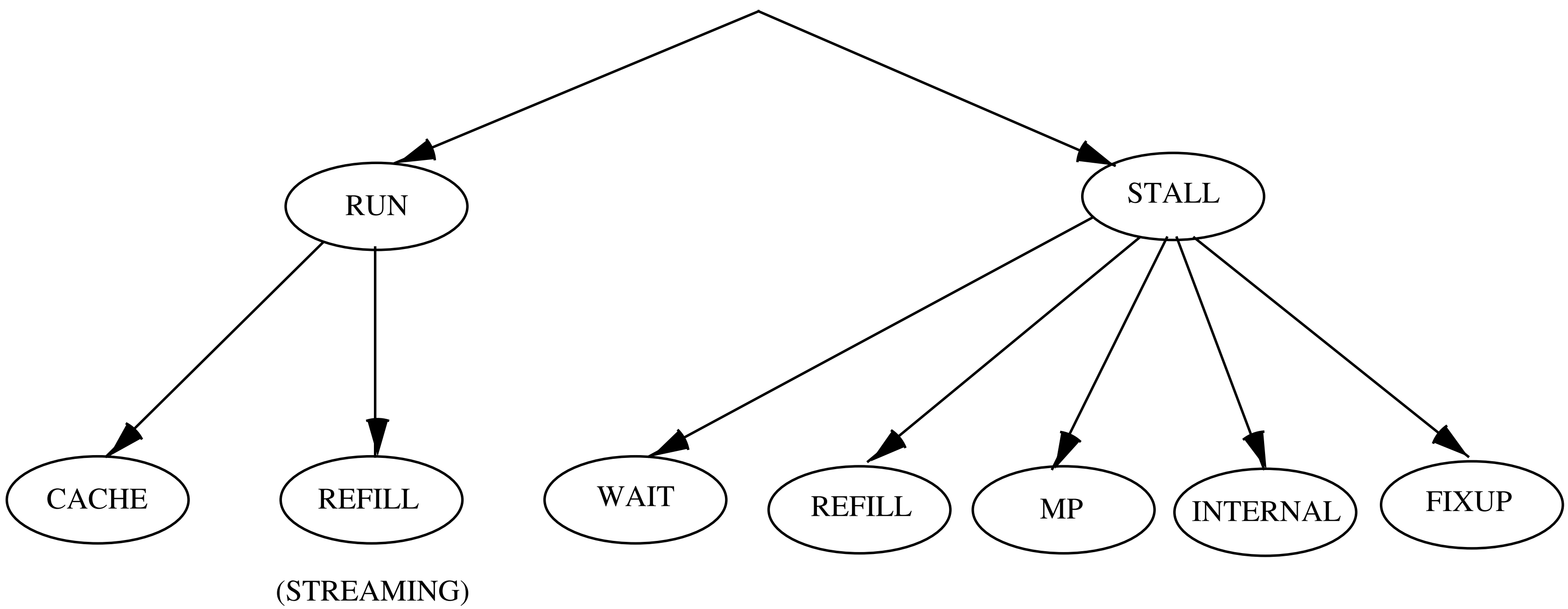
- Normal TLB Access Takes 1/2 Cycle
- Only 1/4 Cycle Available for Translation
- Use TLB for Instructions (Allows 1/4 Cycle Access on Hits)

TLB MISS ST AL



- This Stall Requires no Fixup Cycle

LR3000 CYCLE TYPES



- RUN: Cycle in Which an Instruction Completes
- STALL: Cycle in Which no Instruction Completes

LR3000 CYCLE TYPES

RUN

Cache:	Processor Executes from ICache
Refill (Streaming):	Processor Executes as ICache is Being Refilled

STALL

Wait:	No Cache Activity; Waiting for Stall Resolution
Refill:	Data Transferred from Main Memory to Caches
MP:	Memory System can Access Data Cache
Fixup:	Restart Processor and Coprocessor Pipelines
Internal:	TLB Miss, Coprocessor Busy, Multiplier/Divider Busy

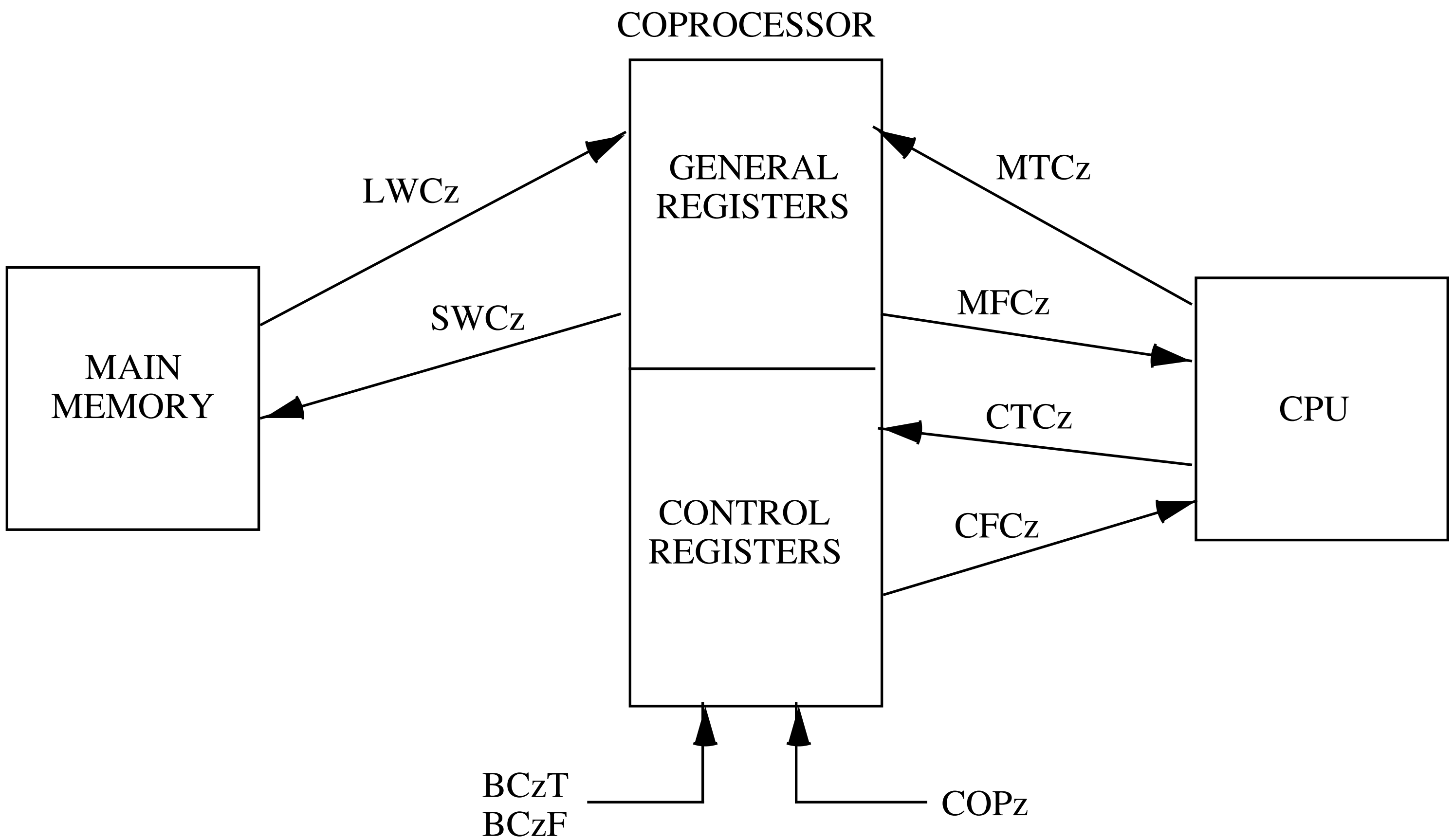
- Streaming is for ICache Refill Only

MULTIPLE STALLS: SERVICE HIERARCHY

1. TLB Miss, Partial Word Store, MP Stalls
2. Data Cache Miss or Write Busy (Mutually Exclusive)
3. Coprocessor Busy
4. Instruction Cache Miss
5. Multiplier/Divider Busy

- Stalls not Requiring Processor Intervention can Self Resolve and not Cause a Stall While Another Stall is Being Serviced

COPROCESSOR INSTRUCTIONS



- z = 0 - 3
- Not All Instructions Supported by All Coprocessors

Register Comparisons

	CPU	FPA
MIPS	32	16 X 64 bits
SPARC	128 (windows 16/proceedure)	32 X 32 bits
AMD	192 (windows variable/proc.)	8 X 64
MOTOROLA	32	(same as integer)
INTEL	32	16 X 64, or 32 X 32

INSTRUCTION SET COMPARISONS

	MIPS	SPARC	MOTOROLA
REGISTERS:			
INTEGER	32 FIXED	WINDOWS	32 FIXED
FP	SEPARATE	SEPARATE	SHARED
BRANCHES:			
CONDITION CODES	NO	YES	NO
COMPARE & BR.	YES	NO	YES
A=B COMP. & BR.	YES	NO	NO
INTEGER MULT/DIV:	HARDWARE	SOFTWARE	HARDWARE