MIPS Instruction Decoding

Dinesh Sharma

EE Department
IIT Bombay, Mumbai

April 19, 2021

Decoding R type instructions

Let us assume that the instruction has already been fetched from the memory.

Type	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
R	6 bits Op type	5 bits Rs	5 bits Rt	5 bits Rd	5 bits shift amt	6 bits Operation

- To implement R type instructions, we must first determine that it is indeed an R type instruction!
- That is easily done – Top 6 bits being 0 indicates that this is an R type instruction. (There are some co-processor related instructions which are also R type – but we shall postpone their implementation to later).
- If the six most significant bits are 0, the specific function to be implemented is determined by the 6 least significant bits.
- These six bits are decoded in two groups of 3 bits each.

Decoding the 6 Least Significant Bits of the Instruction

Type	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
R	6 bits 000000	5 bits Rs	5 bits Rt	5 bits Rd	5 bits shift amt	6 bits Operation

- If the top six bits are 0, bottom six bits are decoded in two sub fields of three bits each.
- Let us first take the group of instructions where the top 3 of these bits (bits 3,4,5) are all zero.
- These are all shift instructions – so the shifter can be activated right away and the operand in Rt is routed to the shifter.
- The actual shifter function is decoded according to the 3 least significant bits (bits 0,1,2).
- If bit 2 is 0, shift amount is constant and is taken from Field 5 (b6-b10). If it is 1, shift amount is taken from register Rs.

Decoding the three LSBs for shift instructions

For instructions where b26-b31 are zero and b5-b3 are also zero, this is a shift instruction:

3LSBs	Function	Assembler instruction
000	Logical Left Shift (const. amount)	sll rd, rt, shift amt.
010	Logical Right Shift (const. amount)	srl rd, rt, shift amt.
011	Arithmetic Right Shift (const. amount)	sra rd, rt, shift amt.
100	Logical Left shift (var. amount)	sllv rd, rt, rs
110	Logical Right Shift (var. amount)	srlv rd, rt, rs
111	Arithmetic Right Shift (var. amount)	srav rd, rt, rs

Since logical and arithmetic left shift are the same, encoding with 01 in the two least significant bits is not used.
b2 = 0 indicates a fixed shift amount (= b11-b6), b2 = 1 indicates a variable amount of shift, taken from register Rs (with index in bits 21-25).

R type instructions with b5-b3 = 001

Instructions with b26-b31 = 000000 and b5-b3 = 001 are jump or call instructions.

3LSBs	Function	Assembler instruction
000	Jump by register (PC = rs)	jr rs
001	Jump and link by register	jalr rd, rs
100	Call a system routine	syscall
101	Implement a debug break point	break

- jalr instruction jumps to the address in register rs, but also copies the return address to rd, typically $ra.
- syscall is system dependent. Typically, function arguments are placed in $ai registers indicating the type of service required (say – print) and then syscall is executed.
- break causes an exception, which is handled by co-processor 0.

R type instructions with b5-b3 = 010

Instructions with b26-b31 = 000000 and b5-b3 = 010 are instructions which move data between the hi and lo registers of the multiplier/divider and MIPS general purspose registers.

- Only 4 of the eight possible combinations are used.
- b2 is always 0 if b26-b31 = 000000 and b5-b3 = 010.

3LSBs	Function	Assembler instruction
000	Move from Hi	mfhi rd
001	Move to Hi	mthi rs
010	Move from LO	mflo rd
011	Move to LO	mtlo rs

R type instructions with b5-b3 = 011

These are multiply/divide instructions. Again, only four of the possible eight combinations is used.
b2 is always 0 if b26-b31 = 000000 and b5-b3 = 010.

3LSBs	Function	Assembler instruction
000	Multiply	mult rs, rt
001	Unsigned Multiply	multu rs, rt
010	Divide	div rs, rt
011	Unsigned Divide	divu rs, rt

The RD field is not used in mult and div instructions. The destination is always to the hi and lo registers.

Notice that b2 is always zero for both groups – 010 and 011 and both groups deal with Hi and lo registers.

R type instructions with b5-b3 = 100

These are the add/subtract and logical function instructions.

3LSBs	Function	Assembler instruction
000	Add registers	add rd, rs, rt
001	Unsigned add registers	addu rd, rs, rt
010	Subtract registers	sub rd, rs, rt
011	Unsigned subtract registers	subu rd, rs, rt
100	Bit wise AND	and rd, rs, rt
101	Bit wise OR	or rd, rs, rt
110	Bit wise XOR	xor rd, rs, rt
111	Bit wise NOR	nor rd, rs, rt

Signed and unsigned versions of addition/subtraction are the same operationally, except that signed operations cause an exception on signed overflow condition.

R type instructions with b5-b3 = 101

These instructions are comparison instructions and only two combinations are used out of the possible 8.

3LSBs	Function	Assembler instruction
010	Set if less than	slt rd, rs, rt
011	Set if less than unsigned	sltu rd, rs, rt

Whether rs is less than rt or not is determined by subtraction, setting rd to 1 if the result is negative. The difference is discarded after this.

Therefore the 3 LSBs are chosen to be the same for slt and sltu as sub and subu. The same decoder is used for bits b2-b0 and bits b5-b3 are used for deciding whether the actual difference or 1/0 goes to Rd.

I type instructions

I type instructions reserve the 16 lsb's for an immediate or an address offset. These have room for a function code and up to two register indices in the top 16 bits.

Type	Field 1	Field 2	Field 3	Immediate
I	6 bits	5 bits	5 bits	16bits
	Op code	Rs	Rt	Immediate
or		Base	data	Addr offset

- The function code is obtained from the top 6 bits.
- The combination with all 0's is used for R type instructions.
- Combinations 000010 and 000011 are used for J type instructions.
- All other combinations indicate I type format.
- The bottom 16 bits are used as immediate data in data operations and as a 16 bit signed offset from the register rs (or pc) when used for address specification.

I type instruction decoding: b31-29 = 000

- The function code (now in the top 6 bits) is decoded in two sub-fields again – b31-29 and b28-26.
- The top 3 bits being zero is used for conditional branches. The immediate field represents a word offset from updated PC.

b28-26	branch condition	Assembler instruction
001	rs $<$ 0	bltz rs, label if rt = 00000
001	rs $\geq$ 0	bgez rs, label if rt = 00001
100	rs = rt	beq rs, rt, label
101	rs $\neq$ rt	bne rs, rt, label
110	rs $\leq$ 0	blez rs, label – rt = 00000
111	rs $>$ 0	bgtz rs, label – rt = 000000

The 16 lsbs with appended 00 and sign extension are added to PC to get the destination address for jumps.
(Pipelined designs use a brach delay instruction after these).

I type instruction decoding: b31-29 = 001

- This group of instructions uses the immediate field as data.
- It is sign extended for signed operations, zero extended for unsigned and logic operations.

b28-26	Operation	Assembler instruction
000	Add Immediate	addi rt, rs, imm
001	Add Immediate Unsigned	addiu rt, rs, imm
010	Set if less than Immediate	slti rt, rs, imm
011	Set if less than Imm unsigned	sltiu rt, rs, imm
100	Bit wise AND with immediate	andi rt, rs, imm
101	Bit wise OR with immediate	ori rt, rs, imm
110	Bit wise XOR with immediate	xori rt, rs, imm
111	Load upper half word with immediate	lui rt, imm

The combination 111 is used for load immediate of upper half word. Therefore Bit wise NOR with immediate is not available.

I type instruction decoding: b31-29 = 001

Notice that R type and I type instructions use the same function code in the lower 3 bits of function code for selecting the same operations.

3 bits	R type	I type
000	Add reg.	Add immediate
001	Add reg unsigned	Add immediate unsigned
010	Subtract reg.	Set if $<$ immediate
011	Subtract reg. unsigned	set if $<$ imm. unsigned
100	AND Reg.	And immediate
101	OR Reg.	OR immediate
110	XOR Reg.	XOR immediate
111	NOR Reg.	load upper half immediate

Subtract immediate is implemented as add immediate with the immediate value changed to its 2's complement by the assembler. The corresponding codes are used for comparison instructions slti and sltiu, which do use the subtractor for their operation.

I type instruction decoding: b31-29 = 100

- These instructions are used for loading registers from memory.
- Sign extended immediate value is added to contents of rs, and the content of memory at this address (byte, halfword or word) is loaded into rt.

b28-26	Operation	Assembler instruction
000	Load byte, sign extend to word size	lb rt, imm(rs)
001	Load halfword, sign extend to word	lh rt, imm(rs)
011	Load word	lw rt, imm(rs)
100	Load byte, zero extend to word size	lbu rt, imm(rs)
101	Load half word, zero extend to word	lhu rt, imm(rs)

I type instruction decoding: b31-29 = 101

- These instructions are used for storing registers to memory.
- Sign extended immediate value is added to contents of rs, and a byte, halfword or word from the data register is stored at this address.
- Since size is not extended in memory, no distinction needs to be made between signed and unsigned quantities.

000	Store byte	sb rt, imm(rs)
001	Store half word	sh rt, imm(rs)
011	Store word	sw rt, imm(rs)

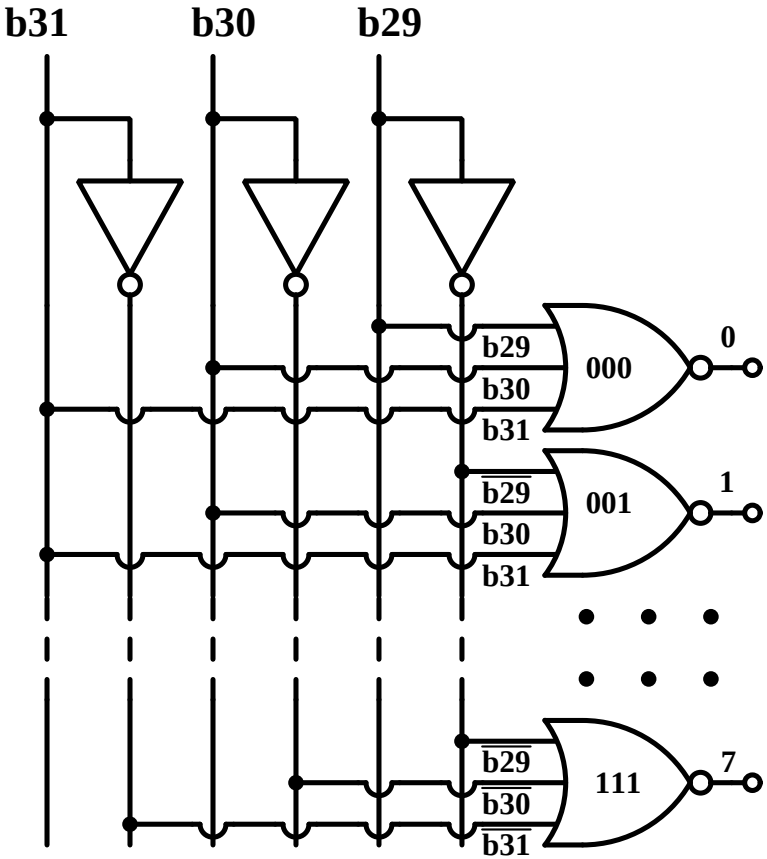
J type instructions

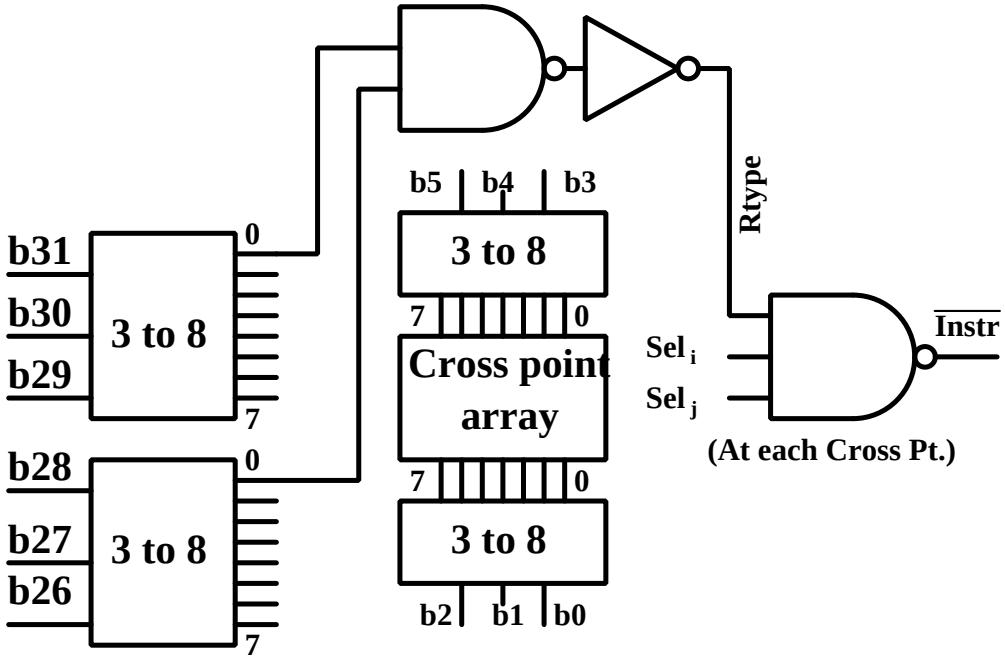
Type	bits 31-26	bits 25-0
J	Op code	Mem. word addr (addr bits 27-2)

- J type encoding is used for instructions j and jal.
- Bit combinations 000010 and 000011 in the funtion codes are used for j and jal respectively.
- 00 is appended to the 26 lsb's of the instruction.
- 4 msb's of incremented PC are added before the above 28 bits to form the jump target address.
- jal instruction (Jump and link) also copies the return address to $ra register.

Circuit Implementation

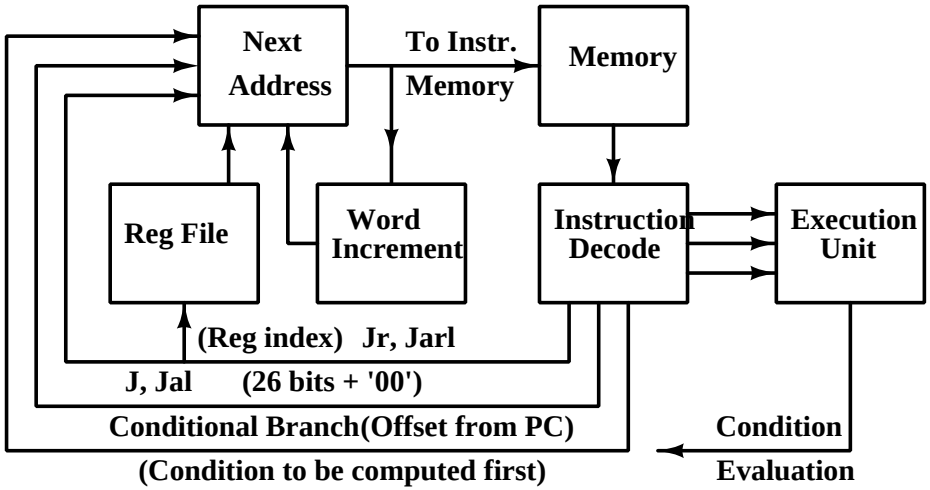
Once the symmetries of instruction code have been established, it is easy to design circuits to decode these.





The circuits above show how the R type instructions can be decoded.

MIPS Control Flow



Next instruction is fetched from:

- Constructed address if J type: (PC_{31-28}&26bits&'00')
- Register content+Offset if Jr or Jarl
- PC + offset if conditional branch
- PC + 4 otherwise.

Next address calculator needs enable signals as well as data (such as offset from PC or register contents) for its operation.

For conditional branching, the condition is to be evaluated first and only then can the offset be added to incremented PC.

Performance Estimation for Single Cycle Design

(Data taken from B. Parhami's slides)

Assume:

Instruction Access	2 ns
register read/write	1 ns
ALU operation	2 ns
Data Cache access	2 ns
Total	8 ns

R type	6 ns	44%
Load	8 ns	24%
Store	7 ns	12%
Branch	5 ns	18%
Jump	3ns	2%
Weighted Mean	$\approx$ 6.36 ns	

We assume that a load occurs through the data cache, while store writes directly to memory. Worst case instruction execution time of 8ns corresponds to a maximum clock frequency of 125 MHz.

Multi-cycle design and Pipelining have the potential of increasing the clock frequency to about 500 MHz.