Goodwill Computer Museum
Relay Computer 3
Instruction Set

Document is an edited portion
of the Relay Paper
written by Harry Porter,
used with permission.

Revision 4 25 Oct 2011

The Instruction Set

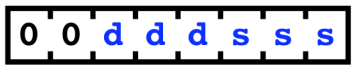
Each instruction is one byte long. Furthermore, several instructions also include two additional bytes, making these instructions three bytes in total. The second and third bytes form a 16-bit value, which is usually used as a memory address.

In the next few sections, we discuss each individual instruction.

The 3 bits of the condition code register (Zero, Carry, Sign) are modified by only the ALU instruction.

MOV-8: The 8-Bit Move Instruction

Move



ddd = destination register
sss = source register
(A, B, C, D, M_1 ,M_2 ,X or Y)

This instruction moves the contents of one of the 8-bit registers to any other register.

If the "source" and "destination" register happen to be the same, this instruction will set that register to zero. In many computers, this function is called the "CLEAR" instruction.

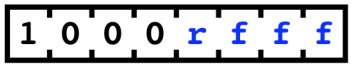
Register Codes (ddd and sss):

000 = A
001 = B
010 = C
011 = D
100 = M1
101 = M2
110 = X
111 = Y

This instruction takes 8 clock periods.

ALU: The ALU Instructions

ALU



r = destination register (A or D)
fff = function code
(add, inc, and, or, xor, not, shl)

This instruction takes, as its input, the current values of the B register and the C register. The result is placed in either the A register or the D register. A single bit in the instruction (called "r") indicates whether to put the result in A or D. The function to be performed is encoded in the 3-bits called "fff".

Destination Register Code (r)

0 = A
1 = D

Function Codes (fff)

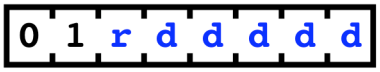
000 = <not used>
001 = ADD
010 = INC (increment B by 1)
011 = AND
100 = OR
101 = XOR
110 = NOT
111 = SHL (shift B left circular 1 bit)

The INC, NOT, and SHL functions use only B and ignore C. If the function code fff is 000 (i.e., <not used>), then zero will be loaded into the destination register. All 3 bits of the condition code register (Zero, Carry, and Sign) will always be updated, regardless of what function code is used.

This instruction takes 8 clock periods.

SETAB: The Load Immediate Instruction

Load Immediate



r = destination register (A or B)
ddddd = value (-16..15)

This instruction can load any value between –16 and +15 into either the A register or the B register.

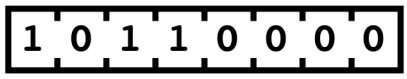
Destination Register Code (r)
- 0 = A
- 1 = B

The 5-bit value (ddddd) is first sign-extended to 8 bits, which allows negative numbers to be specified.

This instruction takes 8 clock periods.

INC-XY: The 16-Bit Increment Instruction

16-bit Increment



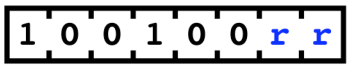
XY ← XY + 1

This instruction increments the XY register.

This instruction takes 14 clock periods.

LOAD: The Load Instruction

Load



rr = destination register (A, B, C, D)
reg ← [M]

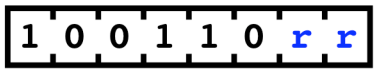
This instruction reads a byte from the SRAM memory and moves it into one of 4 possible registers. The memory address is taken from the 16-bit M Register.

Destination Register Code (rr)
- 00 = A
- 01 = B
- 10 = C
- 11 = D

This instruction takes 12 clock periods.

STORE: The Store Instruction

Store



rr = source register (A, B, C, D)
[M] ← reg

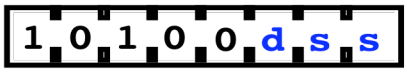
This instruction writes a byte from one of 4 possible registers into the SRAM memory. The memory address is taken from the 16-bit M Register.

Source Register Code (rr)
- 00 = A
- 01 = B
- 10 = C
- 11 = D

This instruction takes 12 clock periods.

MOV-16: The 16-Bit Move Instruction – General Form

16-Bit Move



d = destination register (XY or PC)
ss = source register (M, XY, J or ADDR-SW)

Destination Register Code (d)
- 0 = XY
- 1 = PC

Source Register Code (ss)
- 00 = M
- 01 = XY
- 10 = J
- 11 = front panel address switches

This instruction copies 16 bits from one register to another. Either the XY register or the Program Counter (PC) may be loaded. If the PC is selected as the destination, this will cause a branch in program execution.

If the source register code is 11, then the value set in the front panel address switches will be loaded.

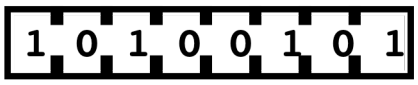
The CALL instruction (discussed below) will branch and save the return address in XY. Thus, by moving XY back to the PC, this instruction can be used as a RETURN instruction.

This instruction takes 10 clock periods.

MOV-16: The Return / Branch-Indirect Instruction

This instruction is a special case of the more general 16-bit move instruction.

Return / Branch Indirect



PC ← XY

This instruction will copy the 16-bit value in the XY register to the Program Counter (PC), which will cause a branch (i.e., a "jump") immediately thereafter.

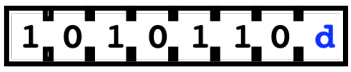
The CALL instruction (which is discussed below) will jump to a subroutine and will also save the "return address" in the XY register. The subroutine, after completing execution, can return by moving the address in XY back into the PC with this instruction.

[Most computers save return addresses on some sort of stack, which permits subroutines to call other subroutines, and even to call themselves recursively. This computer does not directly support a stack. Instead, this computer uses a simpler approach of saving the return address in a register. This approach supports subroutines—an important programming abstraction—as long as they don't call other subroutines. If you wish to call one subroutine within another subroutine, then your program will need to save the return address before calling the second subroutine. A simple STORE and LOAD will suffice if the subroutine is not recursive. By the way, the MIPS microprocessor also uses this approach because of its efficiency; many subroutines don't call other subroutines and would be slowed down by having to access memory.]

This form of the 16-bit move instruction can also be used to branch to an arbitrary computed memory address. This would be useful, for example, in implementing jump tables, switch statements, or threaded execution models.

LDSW: The Load Switches Instruction

Load Switches



d = destination register (A, or D)
reg ← DATA-SW

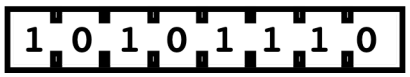
This instruction reads a byte from the settings of the front panel data switches and moves it into one of 2 possible registers.

Destination Register Code (d)
0 = A
1 = D

This instruction takes 10 clock periods.

HALT: The Halt Instruction

Halt

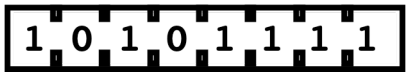


Execution will halt at the completion of this instruction. The program counter will contain the current instruction's location + 1.

This instruction takes 10 clock periods.

HLTRL: The Halt and Reload Instruction

Halt and Reload



PC ← ADDR-SW

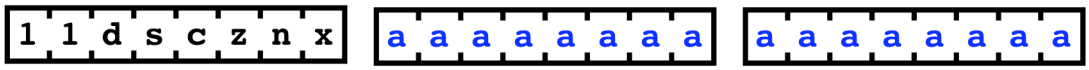
Execution will halt at the completion of this instruction. Furthermore, the PC will be loaded with the value set on the front panel address switches. This is handy to allow restarting a demo or diagnostic subroutine in PROM.

This instruction takes 10 clock periods.

GOTO: The Goto Instruction – General Form

The final instruction class is rather unusual and encompasses several different instruction variants. In the sections following this one, we'll show specific useful examples.

Goto - General Form



Instruction Codes

d: destination (0=M, 1=J)
s: 1=load PC if "sign" bit is set (if negative); 0=ignore sign bit.
c: 1=load PC if "carry" bit is set (if carry); 0=ignore carry bit.
z: 1=load PC if "zero" bit is set (if result is zero); 0=ignore if "zero" bit is clear.
n: 1=load PC if "zero" bit is clear (if result is not zero); 0=ignore if "zero" bit is set.
x: 1=copy PC to XY; 0=do not copy.

This instruction is followed by 2 additional bytes, which contain a 16-bit address value. This value will be loaded into either the M register or the J register. If loaded into the J register, this instruction will branch to the given address. This is useful for its "CALL" and "GOTO" variants. Alternately, the M register can be loaded with a literal 16-bit value, which is especially handy if the next instruction is a LOAD or STORE instruction, which use the M register as a memory address.

This instruction takes 24 clock periods. During the execution of the instruction, the PC is incremented 3 times. The first is the increment that is done for every instruction. The second and third increments are done in conjunction with loading the second and third bytes of the instruction. Afterward, the PC will be left pointing to the instruction following the 16-bit address portion of the instruction.

In the last step of the instruction, the PC will be loaded from the J register, and this load can be done either conditionally or unconditionally. Normally, this instruction is used either (1) to load the M register (in which case the PC is not loaded and no branching occurs) or (2) to branch (in which case the J register is loaded and then the PC is loaded from J).

The purpose of the J register is solely to support branching instructions where the 16-bit target address follows the 8-bit instruction op-code. Memory is always read one byte at a time. During instruction execution, the PC is used to provide sequential addresses while fetching the 16-bit target address. This target address is fetched and placed into the 16-bit J register. Only after the address is complete is the target address moved from J into the PC, causing the jump in the flow-of-control. Furthermore, this final modification of the PC is conditional. Thus, this instruction can be used to implement conditional branching.

The instruction is also capable of copying the PC register to the XY register. This is controlled by the "x" bit in the instruction. If the x bit is set, then the PC will be copied to the XY register at the end of the instruction, thereby setting XY to point to the next instruction following this instruction. This is useful in the CALL variant of the instruction, when a subroutine is invoked. The instruction following the CALL is the return address.

Here is a summary of the actions this instruction may take, in the order they are executed:

Instr = (PC++)
M1 or J1 = (PC++)
M2 or J2 = (PC++)
XY = PC
PC = J (this step is conditional)

The following chart shows the common variants of this instruction. The SETM instruction moves 16-bits into the M Register; the GOTO (also called JUMP) instruction branches unconditionally; Bcond represents several instructions which branch conditionally (as discussed below); the CALL instruction is used to save a return address and jump to a subroutine.

SETM	GOTO	Bcond	CALL	ACTION
√	√	√	√	Instr = (PC++)
√	√	√	√	M1 or J1 = (PC++)
√	√	√	√	M2 or J2 = (PC++)
			√	XY = PC
	√	?	√	PC = J

The final load (from J into the PC) is conditional, based on the values of the condition code register and several tests selected by the instruction. There are four possible tests, corresponding to four bits in the instruction. The bits in the instructions are called s, c, z, and n. If a bit in the instruction is set to 1, the corresponding test of the condition codes is done. If the test is true, the PC is loaded. If the test is false, or if the test is not selected (i.e., the bit in the instruction is 0), then the PC is not necessarily loaded. If any of the tests is true, then the PC is loaded and a branch occurs. If all the tests are either false or not done, then the PC is not loaded.

Here are the 4 bits from the instruction and their meanings:

s (Sign)
1: load PC if the "sign" bit is set (i.e., if result of last ALU instruction was negative)
0: do not test the "sign" bit.

c (Carry)
1: load PC if the "carry" bit is set (i.e., if carry after an ADD or INC instruction)
0: do not test the "carry" bit.

z (Zero)
1: load PC if the "zero" bit is set (i.e., if result is equal to zero)
0: do not test whether the "zero" bit is set.

n (Not Zero)

1: load PC if the "zero" bit is clear (i.e., if result is not zero)
0: do not test whether the "zero" bit is clear.

For example, if only the "s" bit is set, the branch will occur only if the result is negative. Otherwise, the next sequential instruction will be executed. We call this variant "BNEG", for "branch if negative".

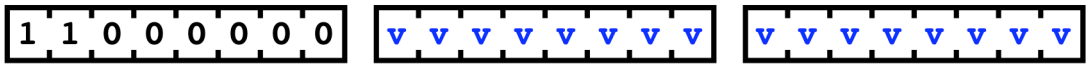
Here are some common test combinations:

s	c	z	n	
0	0	0	0	Never – Used to load the M register with no branching
0	0	0	1	BNZ/BNE – Branch if result not zero; Branch if B!=C after XOR instruction
0	0	1	0	BZ/BE – Branch if result zero; Branch if B==C after XOR instruction
0	0	1	1	JUMP – Used for unconditional branching
0	1	0	0	BC – Branch if carry (useful after ADD or INC)
0	1	0	1	BCNZ – Branch if carry or result not zero
0	1	1	0	BCZ – Branch if carry or result zero
1	0	0	0	BNEG/BLT – Branch if result negative (Branch Less Than)
1	0	1	0	BLE – Branch if result negative or zero (Branch Less Than or Equal)
1	1	0	0	BCNEG – Branch if carry or result negative

GOTO Variant: The SETM Instruction

This instruction is a special case of the more general form of the GOTO instruction.

Load 16-bit Immediate



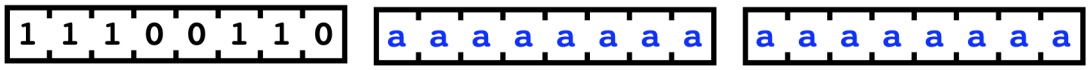
Load the immediate value into M (i.e., M_1 and M_2)

This instruction will load a 16-bit value into the M register (i.e., into the M1-M2 register pair). This instruction is especially useful in conjunction with the LOAD and STORE instructions which expect the M register to contain an address.

GOTO Variant: The JUMP Instruction

This instruction is a special case of the more general form of the GOTO instruction.

Jump



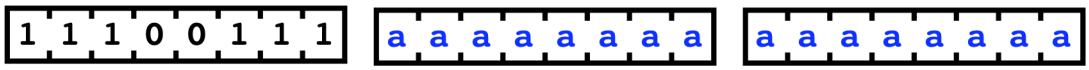
Branch to the given address

This instruction will branch unconditionally to the address given in the instruction. Sometimes this variant is called the GOTO instruction, but the term "GOTO" is used here to refer to the more general form, which includes other variants.

GOTO Variant: The CALL Instruction

This instruction is a special case of the more general form of the GOTO instruction.

Call

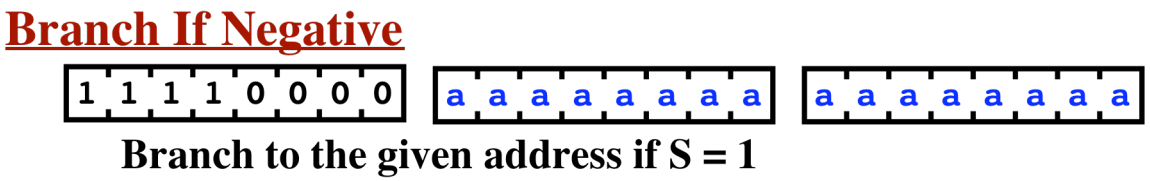


Branch to the given address
Save return location in XY register

This instruction is used to call a subroutine. It will branch unconditionally to the address given in the instruction. In addition, this instruction will save the address of the next instruction in the XY register. Thus, after completing the subroutine, execution can return to the instruction following the Call instruction.

GOTO Variant: The BNEG/BLT (Branch-If-Neg) Instruction

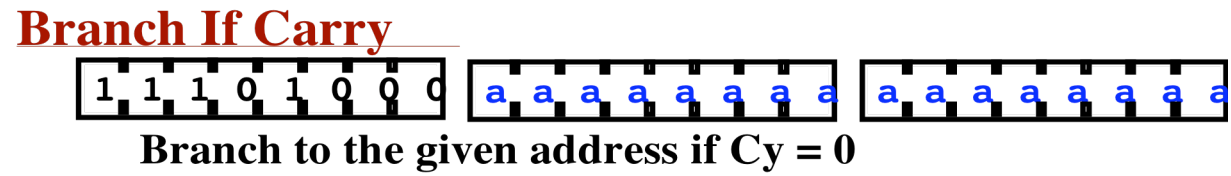
This instruction is a special case of the more general form of the GOTO instruction.



This instruction will branch to the address given in the instruction if the "sign" condition code bit is set to 1. In other words, it will jump if the last ALU instruction (i.e., ADD, INC, AND, OR, XOR, NOT, or SHL) produced a negative result, and not jump otherwise.

GOTO Variant: The BC (Branch-If- Carry) Instruction

This instruction is a special case of the more general form of the GOTO instruction.

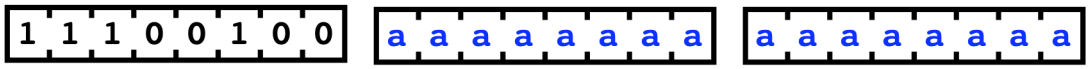


This instruction will branch to the address given in the instruction if the "carry" condition code bit is set (i.e., one). In other words, it will jump if the last ADD or INC instruction resulted in a carry, and will not jump if the instruction did not carry. [Note that the "carry" bit, like the "sign" and "zero" bits, is modified after every ALU instruction, including AND, OR, XOR, NOT, and SHL. The carry bit will always be set from the 8-bit adder circuit, reflecting whether there would have been a carry if B and C had been added, even if the result selected was not from the 8-bit adder circuit. This is not a nuisance since the programmer would normally test the carry bit only after an ADD or INC instruction.]

GOTO Variant: The BZ (Branch-If-Zero) Instruction

This instruction is a special case of the more general form of the GOTO instruction.

Branch If Zero



Branch to the given address if Z = 1

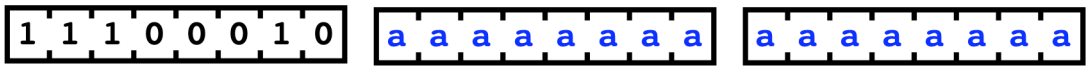
This instruction will branch to the address given in the instruction if the "zero" condition code bit is set to 1. In other words, it will jump if the last ALU instruction (i.e., ADD, INC, AND, OR, XOR, NOT, or SHL) produced a result of zero, and not jump otherwise.

Note that this instruction and the next instruction can be used to test equality. If two values are combined using the XOR instruction, the result will be zero if and only if they are exactly equal. Thus, this instruction is sometimes called the "BE" (i.e., Branch-If-Equal) instruction.

GOTO Variant: The BNZ (Branch-If-Not-Zero) Instruction

This instruction is a special case of the more general form of the GOTO instruction.

Branch If Not Zero



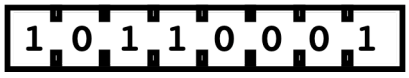
Branch to the given address if Z = 0

This instruction will branch to the address given in the instruction if the "zero" condition code bit is clear (i.e., is zero). In other words, it will jump if the last ALU instruction (i.e., ADD, INC, AND, OR, XOR, NOT, or SHL) produced a result that was not zero, and not jump otherwise.

Since this instruction can be used in equality tests, it is sometimes called the "BNE" (i.e., Branch-If-Not-Equal) instruction.

PRINT: The Print Instruction

Print



This instruction sends a Print pulse to the Robot Typer interface. The character printed is determined by the contents of the B register. See the Robot Typer code table for details.

This instruction takes 8 clock periods.