Description for OSIAC 5362

The Ohio State University, ECE Dept.
Last Updated in July 2016

All words are 16-bit long with the least significant bit labeled 0 and the most significant labeled bit 15.

This machine has four general purpose registers although some have dedicated functions. Each register has two names.

R0 = AC	Accumulator	
R1 = X	Index register	
R2 = SP	System stack pointer	
R3 = PC	Program counter	

Each register is actually part of a high speed memory chip and therefore none of the registers have local increment or decrement control lines. All incrementing and decrementing is done through the ALU system.

This machine has four condition code (CC) bits. Their names and usual meaning are:

C	Carry	Carry-out for add; borrow (complement of carry-out) for subtract
V	Overflow	Overflow for signed 2's complement sense
Z	Zero	Z is true if result is zero
N	Negative	N is true if result is negative

Instructions consist of several types, namely:

- Double Operand Instructions
- Single Operand Instructions
- Branch Instructions
- Special Instructions

Double and single operand instructions, and some of the special instructions use the same group of addressing modes. Instructions are one or more words long. Some addressing modes add words to the instruction after the opcode word.

ADDRESSING MODES

Mode	Name	Assembler Syntax	Operand Is for Arithmetic Instr. (Jump type instr. use EA!)
0	Register	Rk	[Rk]
1	Register Indirect	(Rk)	[[Rk]]
2	Autoincrement	(Rk)+	[[Rk]] then [Rk]+1 $\rightarrow$ Rk
3	Autodecrement	–(Rk)	[Rk]–1 $\rightarrow$ Rk then [[Rk]]
4	Index	n(Rk)	[n+[Rk]]
5	Absolute	n	[n]
6	Immediate	#n	n

Note: Rk is one of the general purpose registers, and n is the contents of an additional word required for the instruction.

INSTRUCTIONS

Double Operand Instructions

Double operand instructions have the following format:

OP2	SAD	DAD	S	D

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

where

    OP2   is the opcode

    SAD   is the source addressing mode

    DAD   is the destination addressing mode

    S   is the source register number, and

    D   is the destination register number.

Let src and dst be the source and destination operands as determined from the addressing modes. Each instruction includes the condition code bits affected.

OP2 Code	Mnemonic	Name	Operation/CC's Affected
1	ADD	add	$[src]+[dst] \rightarrow dst$ N,Z,V,C
2	SUB	subtract	$[dst]-[src] \rightarrow dst$ N,Z,V,C[1]
3	MOVE	move	$[src] \rightarrow dst$ N,Z;V=0,C=0
4	EXG	exchange	$[dst] \leftrightarrow [src]$ CC's unchanged
5	OR	logical OR	$[src]\,OR\,[dst] \rightarrow dst$ N,Z;V=0,C=0
6	AND	logical AND	$[src]\,AND\,[dst] \rightarrow dst$ N,Z;V=0,C=0

---

[1] C indicates a borrow for any subtraction operation. Set to 1 if *no carry-out* from $[dst]+\overline{[src]}+1$ occurs in a single addition step; otherwise, clear to 0. This applies only to SUB, not to any other instructions like DEC.

Single Operand Instructions

Single operand instructions have the following format:

0000	OP1	DAD	QQ	D

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

where

OP1	is the opcode
DAD	is the destination addressing mode,
QQ	is the immediate operand for the ADDQ and SUBQ instructions, and
D	is the destination register number.

OP1 Code	Mnemonic	Name	Operation/CC's Affected
1	CLR	clear	$0 \rightarrow$ dst; N=0,Z=1,V=0,C=0
2	INC	increment	$[dst]+1 \rightarrow$ dst; N,Z,V,C
3	DEC	decrement	$[dst]-1 \rightarrow$ dst; N,Z,V,C
4	NEG	negate	$0 - [dst] \rightarrow$ dst; N,Z,V,C
5	COMP	complement	$\overline{[dst]} \rightarrow$ dst; N,Z;V=0,C=0
6	JMP	jump	$EA(dst) \rightarrow PC$
7	JSR	jump to subroutine	$[SP]-1 \rightarrow SP$ $[PC_{UPD}] \rightarrow [SP]$ $EA(dst) \rightarrow PC$
8	TST	test	$[dst] - 0$; N,Z;V=0,C=0 (Do not cause destination to be written back into memory even though it would seem to be harmless!)

Note: the contents of PC_{UPD} is the address of the instruction following the JSR instruction.

Conditional Branch Instructions

6 instructions: BEQ (Branch on Zero), BNE (Branch on NOT Zero), BPL (Branch on PLUS), BMI (Branch on MINUS), BLS (Branch on Lower than or Same), BHI (Branch on Higher than)
  • Note: BLS and BHI perform UNSIGNED comparisons since they rely on Carry

Branch instructions are 2 words and use a type of relative addressing.

0000	0000	10	I	S/C	CVZN	Offset

First Word | Second Word

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | 15 0

If $\{IR_5 \oplus [IR_3 \cdot (IR_4 \odot C) + IR_2 \cdot (IR_4 \odot V) + IR_1 \cdot (IR_4 \odot Z) + IR_0 \cdot (IR_4 \odot N)]\}$ is true,
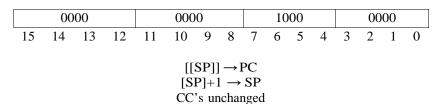
then $[PC_{UPD}] + \text{Offset} \rightarrow PC$

where PC_{UPD} is $PC_{ORIGINAL} + 2$ for this instruction.

Specifically, the encoding of the 6 conditional branch instructions is as follows. As shown above, they are all in the binary format of *0000 0000 10xx xxxx*. This table lists how the last 6 bits are determined for each instruction:
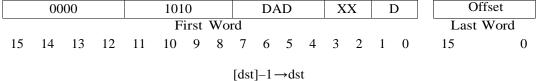
Instruction	IR5 (I)	IR4 (S/C)	IR3 (C)	IR2 (V)	IR1 (Z)	IR0 (N)	opcode
BEQ	0	1	0	0	1	0	0x0092
BNE	0	0	0	0	1	0	0x0082
BMI	0	1	0	0	0	1	0x0091
BPL	0	0	0	0	0	1	0x0081
BLS	0	1	1	0	1	0	0x009A
BHI	1	1	1	0	1	0	0x00BA

Special Instructions

- RTS – return from subroutine

0000	0000	1000	0000
15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

$[[SP]] \rightarrow PC$
$[SP]+1 \rightarrow SP$
CC's unchanged

- DBRA – decrement and branch. Note that this instruction could have 2 or 3 words, depending on the addressing mode. So "Last Word" means either the 2nd word if it has 2 words, or the 3rd word if it has 3 words. See the footnote on Page 4 for the definition of PC$_{UPD}$

0000	1010	DAD	XX	D		Offset
	First Word					Last Word
15 14 13 12	11 10 9 8	7 6 5	4 3 2	1 0		15 0

$[dst]-1 \rightarrow dst$

If $[dst] \neq -1$, then $[PC_{UPD}] + \text{Offset} \rightarrow PC$
CC's unchanged

- SC/CC – set/clear ONLY ONE of the 4 condition status codes: CVZN. Note that one CANNOT set/clear more than one condition code in a single SC/CC instruction.

0000	0000	001	S/C	CVZN
15 14 13 12	11 10 9 8	7 6 5	4	3 2 1 0

S/C = 1/0 for set/clear

- HALT – halt

0000	0000	0000	0000
15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

This instruction halts the processor.

CONTROL LINES

All control lines are labeled on the datapath diagram. There is a pattern to most names.

- I in front of a register name latches a value into that register.
- O in front of a register name makes output of that register available. Note that not all registers have both control lines. See the datapath diagram. The registers AC, X, SP, and PC *do not* have control lines of this form.
- OA - If asserted, T1 is presented to the A side of the adder; otherwise 0.
- IB - If asserted, presents bus logical value to the gates' output; otherwise 0.
- COMP - If asserted, cause 1's complement on all bits; otherwise same value is output. (Programmable inverter!)
- P1 - Causes 1 to be added to the adder.
- OADDER - Latches sum into Q.
- READ - Causes a memory read.
- WRITE - Causes a memory write.
- HALT - Causes the processor to halt.
- This machine has a special set of control lines for accessing the general purpose registers.

  - RAC and WAC (2 bits each) control the selection of the registers for reading and writing, respectively.
    - If RAC/WAC has the value 0, then a read/write is not done to the general purpose registers at all. (Recall that if a control line is not mentioned, it is assumed to be 0.)
    - If RAC/WAC has the value 1, then the selected register is the value of the 2-bit control lines RN/WN.
    - If RAC/WAC has the value 2, then IR bits 3 and 2 (source register number) are used to select the register.
    - If RAC/WAC has the value 3, then IR bits 1 and 0 (destination register number) are used to select the register.

  - For example, to do the register transfer [PC] → MAR, the control lines should be selected as RAC=1, RN=3, IMAR. To do the transfer [R(IR_3:IR_2)] → Q, the control lines should be RAC=2, IQ.

- For updating Condition Code bits, there are the following sets of lines:
    - NEWC causes the cout line (carry-out from the adder) to be saved in C.
    - NEWV causes the vout line (signed 2's complement sense overflow) to be saved in V.
    - NEWZ causes busz (that there is a zero on the bus) to be saved in Z.
    - NEWN causes busn (that there is a negative value on the bus) to be saved in N.
    - SETC/CLRC sets/clears the C condition code. SETV/CLRV sets/clears the V code. OSIAC has no SETZ/CLRZ or SETN/CLRN. You have to do those yourself by putting a zero/negative number on the bus and then NEWZ or NEWN.

There are no other control lines which you can use to cause operations to occur.

CONDITION INPUTS

For conditional transfers a number of conditions are available:

ir1512	ir109	ir8	ir108	ir65	ir11
ir4	ir64	ir118	ir158	ir3	ir7
ir2	ir1	ir0	ir76	ir75	ibrch
cout	vout	busz	busn	ir5	
v	z	n	c		

For example, ir1512 are bits 15:12 of the IR.

ibrch is a condition to help in branch instructions:

$$ibrch = [IR_3 \cdot (IR_4 \odot C) + IR_2 \cdot (IR_4 \odot V) + IR_1 \cdot (IR_4 \odot Z) + IR_0 \cdot (IR_4 \odot N)]$$

The condition cout is the carry-out of the adder; vout is overflow in a signed 2's complement sense; busz is true when the value on the bus is zero; and busn is true when there is a negative value on the bus.

MISCELLANEOUS

Stack convention is the same as the 68000: SP points to the top of the stack and new values are pushed into decreasing addresses.

All registers are 16 bits long except the condition codes C, V, Z, and N which are each 1 bit long. Only T1, T2, T3, T4, T5 and Q can be used as scratch registers.

Maximum limits for this problem are printed by the simulator.

PRINCIPLES FOR IMPLEMENTATION

1. The PC value within an instruction is based on the principle that as soon as a word of the instruction is read, the PC should be immediately updated to point to the next word.

2. Source address calculation is done before destination address calculation.

3. Implement the microcode so that a user's program could run from ROM assuming that they don't try to make the program self–modifying and that all data is accessed from read–write memory.

4. Do not do any unnecessary memory writes. However, redundant memory reads may help make your code more modular and therefore are not a problem. In particular for instructions which never modify the destination, do not write the destination back to itself.

5. Your microcode is not being graded on efficiency. It can't be so grossly inefficient that it can't run for the resources given. My test programs are sometimes 20 statements long and they must run in the resource provided. You should write the code in such a way to optimize your probability of being able to debug it to make it run correctly.

6. How many programs work the first time without any debugging? Not many. Consequently you should be prepared to spend a sizable percentage of your effort in writing test programs which exercise your microcode.

OSIAC 5362 DATA PATHS

AC (R0)   X (R1)   SP (R2)   PC (R3)

↑   ↓

WAC — Combinational Circuit — RAC

WN — — RN

memory interface

O.C. Bus

IIR

IR

IT1   OT1

T1

IB

OA

COMP

IMAR

MAR

SETC
CLRC
NEWC

LD
C   cout

P1

IMDR

MDR

NEWV

OMDR

LD
V   vout

OADDER

IQ

Q

OQ

→
READ

——— ibrch

←
WRITE

IT2   IT4

T2   T4

OT2   OT4

——— HALT

IT3   IT5

T3   T5

OT3   OT5

NEWZ

LD
Z   busz

NEWN

LD
N   busn

9