

Description for OSIAC 662

ECE 662, Autumn 2007

All words are 16 bits 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

| <u>Mode</u> | <u>Name</u> | <u>Assembler Syntax</u> | <u>Operand Is for Arithmetic Instr.</u> <u>(Jump type instr. use EA!)</u> |
|-------------|-------------------|-------------------------|--|
| 0 | Register | Rk | [Rk] |
| 1 | Register Indirect | (Rk) | [[Rk]] |
| 2 | Autoincrement | (Rk)+ | [[Rk]] then [Rk]+1 → Rk |
| 3 | Autodecrement | -(Rk) | [Rk]-1 → 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 *scr* and *dst* be the source and destination operands as determined from the addressing modes. Each instruction includes the condition code bits affected.

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

¹C 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, cleared to 0.

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

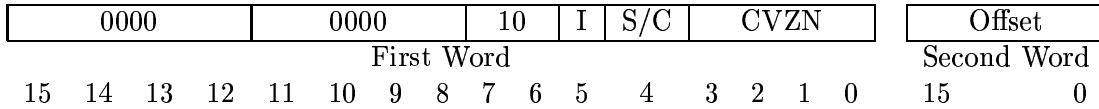
- OP1 is the opcode
- DAD is the destination addressing mode,
- QQ is the immediate operand for the ADDQ and SUBQ instructions, and
- D is the destination register number.

| <u>OP1 Code</u> | <u>Mnemonic</u> | <u>Name</u> | <u>Operation/CC's Affected</u> |
|-----------------|-----------------|--------------------|--|
| 1 | ADDQ | add quick | $[dst] + QQ \rightarrow dst$ N,Z,V,C |
| 2 | CLR | clear | $0 \rightarrow dst; N=0, Z=1, V=0, C=0$ |
| 3 | JMP | jump | $EA(dst) \rightarrow PC$ |
| 4 | JSR | jump to subroutine | $[SP]-1 \rightarrow SP$ $[PC_{UPD}] \rightarrow [SP]$ $EA(dst) \rightarrow PC$ |
| 5 | NEG | negate | $0 - [dst] \rightarrow dst; N,Z,V,C$ |
| 6 | NOT | complement | $\overline{[dst]} \rightarrow dst; N,Z;V=0,C=0$ |
| 7 | SUBQ | subtract quick | $[dst] - QQ \rightarrow dst$ N,Z,V,C |
| 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 JMP instruction.

Branch Instructions

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

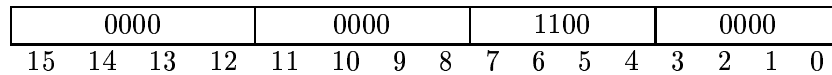


If $\{IR_5 \oplus [IR_3 \cdot (IR_4 \odot C) + IR_2 \cdot (IR_4 \odot V) + IR_1 \cdot (IR_4 \odot Z) + IR_0 \cdot (IR_4 \odot N)]\}$ is true,
then $[PC_{UPD}] + \text{Offset} \rightarrow PC$

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

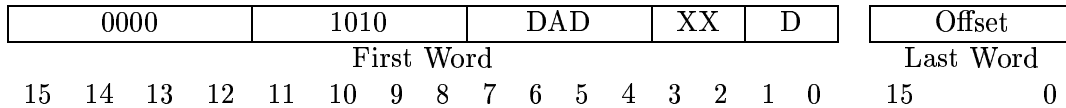
Special Instructions

- RTS – return from subroutine



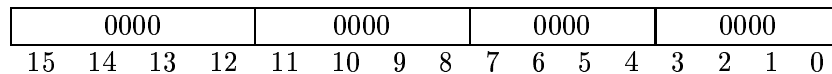
[[SP]] $\rightarrow PC$
[SP]+1 $\rightarrow SP$
CC's unchanged

- DBRA – decrement and branch



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

- HALT – halt



This instruction halts the processor.

ADDRESSING MODES ALLOWED
FOR EACH INSTRUCTION

| Instruction | Addressing mode | Addressing mode | | | | | | | | | |
|---------------------------------|--------------------------------|-----------------|---------|-------------|-----------------|---------------|---------------|----------|-----------|-------|---|
| | | AC | X or SP | (X) or (SP) | (X) + or (SP) + | -(X) or -(SP) | n(X) or n(SP) | Absolute | Immediate | n(PC) | |
| ADD (Add) | s = all d = | x | x | x | x | x | x | x | x | | |
| ADDQ (Add quick) | s = Immed2 d = | x | x | x | x | x | x | x | x | | |
| AND (Logical AND) | s = AC d = d = AC s = | x | | x | x | x | x | x | x | x | x |
| Bcc (Branch conditionally) | | | | | | | | | | | x |
| CLR (Clear) | | x | | x | x | x | x | x | x | | |
| DBRA (Decrement and branch) | | x | x | x | x | x | x | x | x | | x |
| EXG (Exchange) | | x | x | | | | | | | | |
| HALT (Halt) | | | | | | | | | | | |
| JMP (Jump) | | | | x | | | | x | x | | x |
| JSR (Jump to subroutine) | | | | x | | | | x | x | | x |
| MOVE (Move) | s = all d = | x | x | x | x | x | x | x | x | | |
| NEG (Negate) | | x | | x | x | x | x | x | x | | |
| NOT (Complement) | | x | | x | x | x | x | x | x | | |
| OR (Logical OR) | s = AC d = d = AC s = | x | | x | x | x | x | x | x | x | x |
| RTS (Return from subroutine) | | | | | | | | | | | |
| SUB (Subtract) | s = all d = | x | x | x | x | x | x | x | x | | |
| SUBQ (Subtract quick) | s = Immed2 d = | x | x | x | x | x | x | x | x | | |
| TST (Test) | | x | | x | x | x | x | x | x | | |

s – source
d – destination
d₂ – branch destination

cc – indicates condition for branch instruction

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 $[PC] \rightarrow MAR$, the control lines should be selected as RAC=1, RN=3, IMAR. To do the transfer $[R(IR_3:IR_2)] \rightarrow Q$, 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.
 - SETC/CLRC sets/clears the C condition code.
 - 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.

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:

$$\text{ibrch} = [\text{IR}_3 \cdot (\text{IR}_4 \odot \text{C}) + \text{IR}_2 \cdot (\text{IR}_4 \odot \text{V}) + \text{IR}_1 \cdot (\text{IR}_4 \odot \text{Z}) + \text{IR}_0 \cdot (\text{IR}_4 \odot \text{N})]$$

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 662 DATA PATHS

