Instruction Translation

Textbook Section 3.4, 4.8, 6.4

Basics - What We Know

- We know that a CPU executes machine language (ML) instructions.
- We have studied an assembly language, i.e. opcodes and addressing modes. And should be reasonably comfortable with using them to create programs.
- We know that each assembly language instruction (opcode and operands) translates to a corresponding machine language instruction

Basics - What We Do Not Know

What we don’t know is the details of this translation:

- How do assembly language instructions translate to machine language instructions.
- How is the translation process performed.

This lecture focuses on point 1, which is how to:

- translate ASM → ML (“assemble”)
- translate ML → ASM (“disassemble”)

Basics - Ways to Encode ASM in MC

1. Fixed length opcode format
    - In this method the entire instruction: opcode, operands and data size are stored in ONE memory element
2. Variable length opcode format
    - In this method each instruction component is stored in a memory element

Basics

- Intel uses the Variable length opcode format.
- Motorola used the Fixed length opcode format, which we will focus on.
- On the Motorola a ML instruction consists of:
  - an opcode
  - 0-more extension words

Basics - The Opcode

- Is a binary number
- Is a bit pattern
- Uniquely identifies the instruction

The last point means that each instruction is a unique binary number!

Basics - The Opcode

The opcode must specify the:

- operation
- operation size
- number of operands
- type of each operand

Basics - 68000 Opcode

- The 68000 uses 16-bit opcodes.
- There are three general opcode formats:
    - see ref. card under the Status Register

1. `0 - Operand Format`
2. `1 - Operand Format`
3. `2 - Operand Format`

68000 Opcode Formats

0-operand format:

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

All 16 bits are used to encode the opcode

68000 Opcode Formats

1-operand format:

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

effective address

2-operand format:

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

destination source

Filling in the bits requires using the ref. card

Note: reversal of reg/mode in 2-operand destination

ASM $\longrightarrow$ ML

- Each source/destination effective address is specified by a mode/register combination
- On the 68000 Reference Card
    - 2nd & 3rd columns of Effective Addressing mode categories table give the bit pattern for the different modes
- There are **3 bits** for register bits
    - This gives us $2^3 = 8$ possible registers
    - Reason for there being 8 Data Registers and 8 Address Registers

ASM → ML

- 3 mode bits, but more than 8 modes (actually 12)
  - Modes 000-110 require register specification
  - “Mode” 111 requires no register specification, so the “register” bits can be combined with the mode bits to allow 8 more addressing modes
- Not all 6-bit patterns are valid, 3 bit mode + 3 bit register
  - i.e. used reason for illegal instruction error!
- Quick & Implied “modes” have no effective address

Instruction Problem

# “2-operand instructions” > # “0/1-operand instructions”, but fewer bits available to specify instruction.

# of Operands	# of Bits	# of Patterns
0	16	65,536
1	10	1,024
2	4	16

Remember each opcode must be a unique binary number and since the 4 bits are included in the 10 bits and the 10 bits in the 16 bits these patterns are eliminated from the other categories and so in reality there are fewer 0- and 1- operand patterns. But still enough.

Instructions Problem

There are far more operations needing 2 operands than patterns!

See ref. card under “Operation Code Map”. (top of panel with square 68000 chip diagram)

Bits 15-12 specify operation category

Instruction Problem Solution - Part 1

All 0/1-operand instructions (“misc.”) begin with 0100.

- For the 0/1-operand formats this leaves 12 bits in which to specify the instruction, the size & the effective address, which is sufficient.
- This leaves 15 patterns for 2-operand instructions (and branches)

Note: Not all 15 patterns were assigned

  - Bit patterns 1010 or $A and 1111 or $F were reserved for computer designer’s use

Instruction Problem Solution - Part 1

- This leaves 13 patterns, but there are more than 13 2-operand instructions!
- Even worse, up to 3 different sizes (b/w/l) must be supported!

1 size	=	0 extra bits
2 sizes	=	1 extra bit
3 sizes	=	2 extra bits

Instruction Problem Solution - Part 2

68000 designers used “tricks”:

- restrictions placed on some instructions – not all addressing modes & sizes valid (the instruction set is not fully “orthogonal”)
  - more than one way to accomplish a given task **OR**
  - not all registers can be used in all addressing modes
- the bits not required for these items can be used to specify something else in opcode, i.e. different instruction, etc.

Opcode Bit Patterns

- for each instruction are found in column 16 (the third last column) in the Addressing Mode table
- E.g. MOVE instruction (no tricks and no restrictions on move):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	X	X	register			mode			mode			register		
		size		destination						source					

- Notice that this is the pattern for b/w for long the XX is 10.
- To find patterns for b/w go to the bottom of the table in the Opcode Bit Pattern Codes:

Opcode Bit Patterns

- IMPORTANT!! Notice that there is S (Size) and XX (Move Size). When translating the appropriate one must be used.
  - For Move Size → Byte == 01 and Word == 11
  - For Size → Byte == 00 and Word == 01

Translation Example - Move

Currently limit to Dn, An, (An), (An)+ and –(An):

```
    move.w  (a5),d3
```

First check if this instruction is valid – yes it is.

So, the bit pattern is:

```
    00XX RRRM MMee eeee   from the 2-operand format
    00XX RRRM MMmm mrrr   the e's are really
              dest    src
```

XX is the instruction size, which is .w so 11 from legend at bottom

Translation Example - Move

Destination is d3 so:

- addressing mode is data register direct mode
- Using the EA Mode Category table MMM is 000
- And the register is number 3 so RRR is 011
- Source is (a5) which is address register indirect mmm is 010
- And register number 5, so rrr is 101

Translation Example - Move

Now place each of these in the corresponding place in the bit pattern:

```
            dest src
    00 XX RRR M MM mm m rrr
    00 11 011 0 00 01 0 101
```

Converting to hex gives:

```
    3   6   1   5
```

Translation Example - Move

The hex from the previous slide is: `0x3615`

This is the machine code for this instruction. To verify make an assembly program consisting of only this instruction, assemble and invoke the debugger.

In the disassembly window determine the address of the instruction. Then find this address in the memory window and view the word memory contents at this location!

Translation Example - Add

Add has a trick/ restriction: either source or destination must be a data register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	register			X	size		mode			register		
		size		src/dst			opmode			other operand					

Since it is known that the instruction must use a data register the mode bits are not required for this and can be used in an alternative way.

Bit 8 specifies direction:

- 0 = data register is destination
- 1 = data register is source

Translation Example - Add

For ADD opmode bits:

- 000 = data register is destination, size is byte
- 001 = data register is destination, size is word
- 010 = data register is destination, size is long
- 100 = data register is source, size is byte
- 101 = data register is source, size is word
- 110 = data register is source, size is long

For ADDA opmode:

- 011 = address register is destination, size is word
- 111 = address register is destination, size is long

This explains why ADD/ADDA, CMP/CMPA, etc. are different instructions

Quick Instructions - ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Q	Q	Q	0	S	S	E	E	E	E	E	E
											mode			register	

- The immediate value is limited to values between 1 and 8
  - 1 – 7 are 001 to 111
  - In all of the quick instructions it makes no sense to use 0 so 000 is taken as 8

ASM → ML

To assemble:

- look up “opcode bit pattern” for instruction
- fill in:
  - effective address bits
  - size bits
  - etc.
- remember to generate extension words, if necessary (soon to come)
- you can check your work in Devpac

ASM → ML

Exercise: assemble the following – write the 68000 opcode in hex:

```
move.w      a3,d4
add.l       d1,(a0)
nop
clr.b       (a0)+
trap    #1
exg.l       d0,a0
exg.l       a0,d0
```

Solution on the next slide!

ASM → ML

Exercise: assemble the following – write the 68000 opcode in hex:

```
    move.w      a3,d4           380B
    add.l       d1,(a0)         D390
    nop                         4E71
    clr.b       (a0)+           4218
    trap    #1              4E41
    exg.l       d0,a0           C188
    exg.l       a0,d0           C188
```

Invalid operands – so assembler switches to d0,a0