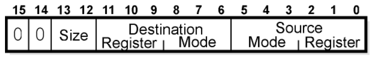
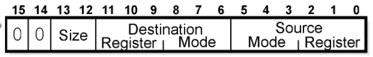
%DVLF ,QVWUXFWLRQV

- Each instruction has an associated instruction template.
- This defines the word data required to encode the instruction (i.e.just fill in the bits).

Move Instruction





Source:	Where to find data
Destination:	Where to copy data
Size field:	

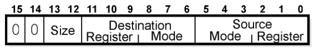
  - 01 -> byte operation
  - 11 -> word operation
  - 10 -> long operation

Register:	Which register to use (if none then used for other purposes.)
Mode:	Defines the data format/location

Copying from Data Register to Data Register

Destination Mode	=	000
Source Mode	=	000

Example : move a word of data from d5 to d3



00 11 011 000 000 101 ⇒ $3605

The instruction is fully self contained
-> no extra operand data is necessary!

([DPS0H 3URJUDP

- Copy value in register d3 to register d0, d1 and d2
- What instruction sequence is required?
  - Move word d3 -> d0
  - Move word d3 -> d1
  - Move word d3 -> d2
- Where do we locate the code in memory? -> $4000

```
4000    30 03    * D0 <- D3
4002    32 03    * D1 <- D3
4004    34 03    * D2 <- D3
```

Copying from Memory to Data Register

->	Source	=	Memory
->	Source Mode	=	111

Source register field encodes memory address length:

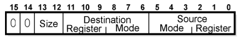
000 if memory address is 16-bit
001 if memory address is 32-bit

Assume all addresses are 32-bit (long-words).

16-bit address used for compact programs and limit the available address space.

([DPS0H

Copy (word) contents of memory at $2000 into register D3



0011 0110 0011 1001 → $3639

This instruction requires an operand = memory address

```
4000    36 39    * Move 2000 ->D3
4002    00 00
4004    20 00
```

Why is the operand 2 words in length?

- We are not moving the value $2000 into d3 register, rather the contents of memory location $2000
- Note:
  - We may not know the contents at this point.
- This is known as **absolute addressing**

([DPSOH 3URJUDP

```
4000    3639    * Move $2000 -> D3
4002    0000
4004    2000
4006    3003    * D0 <- D3
4008    3203    * D1 <- D3
400A    3403    * D2 <- D3
```

This program copies the word at $2000 into D3 and then copies from there into D0, D1, and D2

How is the PC updated?

- Recall the FETCH->DECODE->EXECUTE cycle
- CPU determines the number of operands by looking at the mode fields of the instruction during decoding
- -> It adds 4 to the PC to skip over the 4 byte operand
- Remember the PC is immediately incremented by 2 as soon as the instruction is read in

The Complete Sequence of events

1. **Fetch** instruction at $4000
2. PC <- PC + 2
3. **Decode** instruction
4. PC <- PC + 4
5. **Execute:** Move word from $2000 -> D3
6. **Fetch** instruction at $4006
7. PC <- PC + 2
8. **Decode** instruction
9. **Execute:** Move data from D3 -> D0
10. **Fetch** instruction at $4008
11. PC <- PC + 2
12. **Decode** instruction
13. **Execute:** Move data from D3 -> D1
14. **Fetch** instruction at $400A
15. PC <- PC + 2
16. **Decode** instruction
17. **Execute:** Move data from D3 -> D2

Copying from Memory to Memory

-> Source & Destination = Memory

-> Source & Destination

Mode	Register	
111	000	If memory address is 16-bit
111	001	If memory address is 32-bit

Example:

Move a word from $2002 to $200A:

15	14	13 12	11 10 9	8 7 6	5 4 3	2 1 0
0	0	Size	Destination Register	Destination Mode	Source Mode	Source Register
0	0	11	001	111	111	001

-> $33F9

Memory to Memory Program

```
4000    33F9    * move.w $2002 -> $200A
4002    0000    * Source operand
4004    2002
4006    0000    * Destination operand
4008    200A
```

This instruction requires 2 operands, each 2 words long.

The **source** always precedes the **destination** if multiple operands are required.

The PC will have to be incremented by 8 Bytes (as well as the normal 2 bytes) to point to the next instruction.