ENEE350 **Notes on PDP 11 Addressing** C. Silio

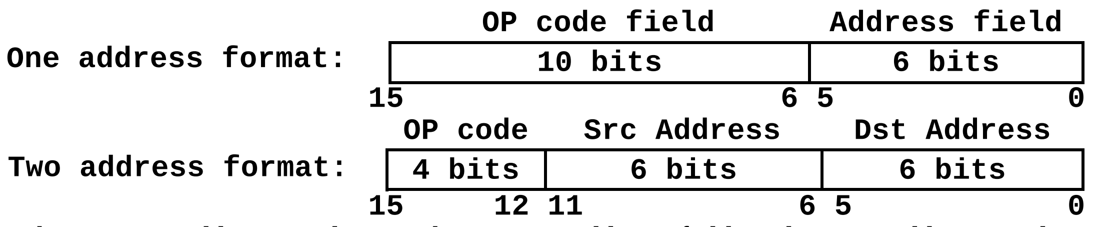
**PDP−11 Addressing:** The PDP−11 has eight CPU registers, R0, R1, ..., R7 that programmers may use or reference. Register R7 is the program counter (PC), and register R6 is the stack pointer register (SP) used implicitly by the Call instruction. Use of the autoincrement and autodecrement addressing modes permit the programmer to use any of the registers as stack pointers and to construct Push and Pop instructions using Move instructions with appropriate source and destination addressing modes.

The PDP−11 is a byte addressable machine with memory organized into 16 bit words (2 bytes); it has a 16−bit data bus to memory, so 2 bytes at a time are fetched from memory. Therefore, the program counter is incremented by 2 for each memory word fetched.

A map of the main memory organization is shown below. Words have even addresses 0, 2, 4, ..., but 8−bit bytes are individually addressable.

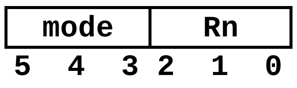
| Contents | | Word addresses |
|---|---|---|
| byte 1 | byte 0 | 0 |
| byte 3 | byte 2 | 2 |
| . | . | . |
| . | . | . |
| . | . | . |
| byte i+1 | byte i | i |
| . | . | . |
| . | . | . |
| . | . | . |
| Byte $2^{16}-1$ | Byte $2^{16}-2$ | $2^{16}-2$ |

The PDP−11 uses expanding op codes and has several distinct instruction formats. The formats for one address instructions and for two address instructions are shown below.



where “Src Address” indicates the source address field and “Dst Address” indicates the destination address field.

Each 6 bit address field (in both the one and two operand instructions) has the following format:



The addressing modes (along with assembly language syntax ) are given in the following table:

**Notation:** Let [Rn] denote the contents of register Rn, and let [[Rn]] denote the content of the (memory) location pointed to by the content of (i.e., the value in) register Rn.

**PDP−11 Addressing Modes**

| Mode bits b5 b4 b3 | Decimal Value | Name of Mode | Assembler Syntax | Meaning |
|---|---|---|---|---|
| 0 0 0 | 0 | Register | Rn | Aeffective = Rn (that is, Operand = [Rn]) |
| 0 1 0 | 2 | Autoincrement | (Rn)+ | Aeffective = [Rn]; then increment Rn |
| 1 0 0 | 4 | Autodecrement | −(Rn) | First decrement Rn; then Aeffective = [Rn] |
| 1 1 0 | 6 | Index | X(Rn) | Fetch X; Increment PC; Aeffective = X + [Rn] |
| 0 0 1 | 1 | Register indirect | @Rn | Aeffective = [Rn] |
| 0 1 1 | 3 | Autoincrement indirect | @(Rn)+ | Aeffective = [[Rn]]; then increment Rn |
| 1 0 1 | 5 | Autodecrement indirect | @−(Rn) | Decrement Rn; then Aeffective = [[Rn]] |
| 1 1 1 | 7 | Index indirect | @X(Rn) | Fetch X; Increment PC; Aeffective = [X + [Rn]] |

Bits b4 and b5 specify the four basic addressing modes. Determination of the effective address of an operand can be viewed as follows. Bits b5 and b4 specify how to obtain an address value. If b3 = 0, this value is the effective address of the operand, denoted Aeffective in the table; but if b3 = 1, the value obtained is the address of a (memory) location that contains the effective address. Thus bit b3 is used to designate indirect addressing. In summary then:

<u>Register mode:</u> When b3 = 0, the operand is the contents of register Rn. In the indirect version of this mode, that is when b3 = 1, register Rn contains the effective address of the operand.

<u>Autoincrement mode:</u> The effective address is in Rn. After the contents of Rn have been used to fetch the operand, Rn is automatically incremented (note that this is standard computer jargon meaning that the contents of Rn are incremented). If the operand is a byte, as specified in the instruction's OP−code field, register Rn is incremented by 1. If the operand is a word, Rn is incremented by 2. In the indirect case, the effective address is contained in the memory location pointed at by Rn before incrementing is performed, and Rn is always incremented by 2.

<u>Autodecrement mode</u>: The contents of Rn are decremented and then used as the effective address. For byte operands, Rn is decremented by 1 (again, jargon meaning the contents of Rn are decremented by 1); for word operands, Rn is decremented by 2. In the indirect version, the effective address is contained in the memory location pointed at by Rn after it has been decremented, and Rn is always decremented by 2.

<u>Index mode:</u> The effective address is generated by adding the contents of Rn to the constant X, which is contained in the word immediately following the instruction word containing the OP CODE. If indirection is specified, it is performed after indexing.

An example of indexed addressing shown below. Note that the instruction occupies two consecutive words in memory. Before the instruction is fetched into the CPU the PC (R7) contains the value i and therefore, points at the first word of the instruction. After this word is fetched into the IR, the PC is incremented by 2 to point to the second word at memory location i+2, which contains the value X. When the OP Code and mode bits are decoded, indicating indexed addressing, the value X in word i+2 is fetched into the CPU and the PC is again incremented by 2, thus pointing at the first word of the next instruction in location i+4. The microprogram interpreting this instruction then adds the contents of register Rn to the value X to generate the effective address of the operand. If Rn contains the starting address of a list of data items, then the index mode can be used to access an individual item by using X as its displacement from the starting address. Another important use for index mode is where the program counter R7 is used as the index register, in which case this mode is referred to as "relative mode". In relative mode X represents a displacement relative to the current value in the PC when it is used to reference an operand.

The following table illustrates one use of indexed addressing in two operand format instructions. Note that mode 6, indicating indexed addressing, can occur in either the source or destination address field . Any of the other addressing modes can be specified in the other field. In the example shown mode 0, register addressing, is specified in the destination address field. Below the table we show how the source address effective address is calculated.

| | Address | Contents | Mode | Src Reg | Mode | Dst Reg |
|---|---|---|---|---|---|---|
| Current | i | OP CODE | 6 | Rn | 0 | Rk |
| Instruction | i+2 | X | | | | |
| Next Instruction | i+4 | | | | | |
| | . | . | . | . | . | . |
| | . | . | . | . | . | . |
| | . | . | . | . | . | . |

**Src_Aeffective = X + [Rn]**

An example of the assembly language mnemonic for indexed addressing is: MOV 25(R3), R5 wherein the content of the memory location whose effective source address = 25+[R3] is copied into register R5 (the effective destination address). In this case word i would contain $016305_8$ and word i+2 would contain $000031_8 = 25_{10}$.

Some basic two operand instructions are summarized below using the following notation; let

| | |
|---|---|
| β | β = 0 for word and β = 1 for byte instructions |
| MSB | Most significant bit (i.e., leftmost bit) |
| dst | destination effective address |
| src | source effective address |

| Mnemonic (Name) | OP code (octal) | Operation performed |
|---|---|---|
| **MOV{B}** (Move) | **β1** | **dst <-- [src] ; In MOVB, if dst is a register, then MSB of the low order byte is extended into the high-order byte.** |
| **ADD** (Add) | **06** | **dst <-- [src] + [dst]** |
| **SUB** (Subtract) | **16** | **dst <-- [dst] - [src]** |