CS 320
Ch. 13 Instruction Sets: Addressing Modes and Formats

There are 7 generic addressing modes depending on how you count:

Immediate
Direct
Indirect
Register
Register Indirect
Displacement
Stack

The *effective address* in a system without virtual memory is the a**ddress of main memory or a register.**

The *effective address* in a system with virtual memory is t**he address is a virtual (logical) address or a register.**

Note that paging is invisible to the programmer and has no effect on the addressing mode.

Formally, the addressing mode is the m**ethod whereby the operand specifies the data.**

The addressing mode is specified in one of two ways:
A) it can be part of the op code in which case each new addressing mode for each instruction needs another op code;
B) It can be a separate mode field.

Immediate addressing - Operand is the data.
Adv: no memory reference is necessary to get the data.
Disadvantage: size of data field is limited to size of address field in the instruction.
Direct addressing - Operand is the memory address of the data.
Adv: only one memory reference and is very simple.
Disadvantage: Provides a limited address space by today's standards.
Indirect addressing - **Operand is the address of a memory location that holds the address of the data.**
Adv: Since address of the data is in memory, it can be larger.
Disadvantage: Two memory references are needed and at times this instruction could cause two page faults.
Register addressing - **Similar to direct but operand is the address of a register that holds the data.**
Adv. Since there are few registers the operand can be small. In addition, there is no memory reference needed for the operand itself.
Disadvantage: The address space is limited to the number of registers that are available.
Register indirect addressing - **Similar to indirect but operand is the address of a register which holds the memory address of the data.**
Adv: Uses one less memory reference than indirect addressing.

Disadvantage: Limited size of address field.

Displacement addressing: **Operand is added to a register. This sum is the memory address of the data.**

Adv: Provides an easy method for iterative processes.

Disadvantage: somewhat complex and slow.

The author lists three kinds of ***displacement addressing***. **Relative addressing, base-register addressing, and indexing.**

Relative addressing. **Operand is added to the program counter register to get the location of a jump target.**

Base-register addressing - **The operand is considered to be an offset which is added to a base register to get the memory address of the instruction.**

Indexed addressing - **The operand is considered to be a memory address which is added to a register which holds a displacement to form the memory address of the data.**

Note that there is a subtle difference between base-register addressing and indexed addressing. **The operations and the result is the same but in practice an indexed instructions often have more bits available in the operand since the operand is considered to be a memory address and not just an offset. Also, some machines use both modes in order to combine them to get based-indexed addressing.**

Autoindexing – with regard to an addressing mode the index register is automatically incremented or decremented before or after the indexed operation so as to be ready for the next iteration.

Stack addressing **- is a form of implied addressing in which no operand is needed. The stack pointer holds memory address of the data and is automatically updated after each operation so as to be ready for the next.**

The Pentium system has a number of fairly complex addressing modes. Two of these are *Scaled indexed addressing* and *based scaled index with displacement*.

Scaled indexed addressing - Index register is shifted (multiplied by 1, 2, 4, or 8) before being used as an index register.

Based scaled index with displacement - The index register is scaled, added to a base register and that sum is added to the operand to get the address of the data.

The ARM processor uses "Load and Store Addressing". **Load and Store are the only instructions to reference memory.**

Preindex and *postindex* with respect to load and store addressing on the ARM processor have the following meaning:

In preindex the base register is added to the offset and this sum is the effective address.

In postindex the base register has the effective address. The base register is added to the offset to update the base register after addressing.

The ARM processor also has a Load/Store Multiple mode. **In this mode a range of registers can be loaded or stored in memory.**

The term *instruction format* d**efines the layout of the bits of an instruction, in terms of its constituent parts.**

There are a number of design tradeoffs in determining the instruction length with regard to s**peed and space.**

Longer instructions mean more (possibly wasted) space but they can often do things that shorter instructions cannot. Longer instructions provide more op codes, more operands, and a richer set of addressing modes. They do so at the expense of space.

It important that the instruction length be a multiple of a character length and a multiple of the length of fixed-point numbers. Characters and fixed-point numbers need to fit into a memory word. Otherwise there would be a lot of wasted bits. A memory word is the basic unit of transfer of data between memory and the CPU. So if an instruction length is not a multiple of the word length, instructions straddle word boundaries and multiple accesses are needed for instructions with possible wasting of bits.

For a given instruction length there are design tradeoffs between the number of op codes and the power of the addressing capability of the instruction. **More op codes means more bits in the op code field. This reduces the number of bits available for addressing.**

Some machines use a *variable length op code.* **Op codes don't have to be of fixed length. Short op codes can be allocated to instructions that need more addressing capability and longer op codes can be allocated to instructions that have implied addressing or less of a need for extensive addressing.**

The number of addressing modes available can alter the instruction length. **More addressing modes implies more bits to determine which addressing mode is being used.**

There are design tradeoffs between registers and memory: **For 2^n registers you need n bits for an address. These n bits come out of the instruction length. Typical instructions have two operands. So 2*n bits in an instruction can address a source and destination operand from 2^n registers. The more registers, in general, the fewer memory accesses are necessary for processing data. The fewer registers the shorter the instruction and the faster the instructions can be fetched.**

The accumulator architecture makes an accumulator part of every arithmetic or logical instruction. There are advantages and disadvantages to this architecture.

Adv: very short instructions since the accumulator is implied as one operand.

Disadvantage: Instructions are awkward and the accumulator is often overused. It typically takes several instructions to do simple operations that could be done in a single instruction in a non-accumulator architecture.

In genral, 32 registers are deemed adequate in contemporary architectures.

A ***register set*** can have **multiple groups of registers in their own address space. For example, the Pentium has data registers and segment registers in two sets. The 8051 has 32 8-bit registers in 4 banks (sets). This makes the addresses shorter but still allows a large number of registers.**

Address granularity **- is a measure of how small of a unit the address can get to. Most machines are byte addressable but some are only word addressable and some small machines are bit addressable.**

Orthogonality as it applies to computer architecture **is a measure of the independence the various fields in an instruction have from one another. For example, if an instruction (op code) can use any addressing mode and any register for all operations then there is a high degree of orthogonality. If, on the other hand, op codes imply a specific addressing mode or some instructions are limited to certain registers, then there is less orthogonality.**

Completeness with respect to computer architecture **is a measure of whether or not each arithmetic data type has an identical set of operations. Thus if a machine can only do multiplication on bytes and not on words it is deemed less complete.**

A computer architect can decide between fixed length instructions and variable length instructions.

F**ixed length instructions are simpler and lend themselves to a simpler pipeline and CPU structure.**

Variable length instructions are more complex but provide more flexibility. Variable length instructions typically provide a more compact and efficient instruction set from a programmers point of view.