Memory addressing

Simon Hollis, COMS12200

What is addressing?

- When we wish to access memory (as opposed to registers), we need to specify which memory address to use
  - e.g. MEM[10] -- access memory address 10
- Ideally, we could directly specify a memory address every time, but this is not always possible.
- Sometimes, we would actually like to specify a *sequence* of addresses.
- Therefore, we have invented many different ways to specify a memory address.

What's in an address?

- The notion of an address is a mechanism for finding information.
- Sometimes this is very easy in that it is already supplied to us.
- For example, we think of constant supply as a form of addressing. You've seen it already as the MOV instruction.

Immediate addressing

- ***Immediate addressing*** is when data is supplied in an instruction --- there is no ‘real’ memory address, and all information is embedded in the instruction and data is ‘immediately’ available.
- e.g. ACC <- 42
- Very fast and simple

Immediate addressing

- Immediate addressing is the simplest form of addressing.
- Pros:
  - All information embedded in instruction (good for pipeline)
    - Makes it very fast
  - Easy to understand
  - Good for optimisers to analyse

Immediate addressing

- Cons:
  - Lack of flexibility
  - Must be inserted statically
  - Limited range

Direct addressing

- Earlier in the course, we saw instructions like MEM[10] <- ACC
- How is this instruction actually formulated?
  - Operation | Operand
  - i.e. 2 | 10
- This is called ***Direct addressing***
  - The exact memory address used is embedded in the instruction

Direct addressing

- Direct addressing has the same pros and cons as immediate addressing.
- Pros:
  - All information embedded in instruction (good for pipeline)
  - Easy to understand
  - Good for optimisers to analyse

Direct addressing

- Cons:
  - Lack of flexibility
    - Must be inserted statically
    - Limited range
    - Slower than immediate addressing

Memory-indirect addressing

- ***Memory-indirect addressing*** solves the problem of limited range by storing the address to be accessed in memory itself.
- e.g. **`MEM[ MEM[ 42 ] ] <- ACC`**
  - Meaning: go and look at memory address 42 and *fetch* the value there.
  - That value is the address to *write* the ACC to.

Memory indirect addressing

- Plus point:
  - The source memory location for the address may be dynamically changed.
- Direct addressing still has some drawbacks:
  - The first memory address is still statically compiled.
  - The range restriction now applies to the initial memory range.

Register-indirect addressing

- ***Register-indirect*** (or *Register*, or sometimes *Indirect*) addressing provides much more flexibility.
- Idea: use a register's value as the memory address.
- e.g. **`MEM[ ACC ] <-- X`**
- More commonly used in register machines, e.g. **`MEM[ r1 ] < -- r2`**

Register-indirect

- There are lots of advantages to register-indirect addressing:
  - The memory address can be *dynamically computed*.
  - The value does not need to be stored in the instruction, reducing code size
  - *The register is internal* to the processor -- faster, more energy efficient.

Pointers

- Indirect addressing allows native support of **pointers**, a key programming primitive.
- Accessing indirectly is equivalent to a *de-referencing* operation (e.g. `*p` in C)
- <example>

Indexed addressing

- Sometimes, it makes sense to define a base address and access memory based on this.
  - Useful for stacks, arrays, caches...
    - *Second half of the course will discuss*
  - Indexed addressing extends indirect addressing to support this.
  - We have a ***base*** address and an ***offset***.

Indexed addressing

- Normally, the base and offset are both stored in registers, although this need not be the case.
- We gain instructions like

MEM [r1 + r2] <-- r3

- r1 is the base, r2 the offset
- Base and offset are orthogonal, meaning comes from how they are used.

Indexed addressing

- Many implementations support the base + offset construct natively.
- Architectures often gain a dedicated register to help, normally called either:
  - The stack pointer
  - Or the base register
    - *(and sometimes both!)*

Indexed addressing

- The stack / base registers may or may not be general purpose, depending on the architecture.
- The offset normally comes from an additional register, usually a general purpose one, although some architectures

Example

- <example of index addressing, based on an array>

Summary

- We have seen 5 methods of memory addressing.
- We have evaluated their use and efficiency.
  - Some directly relate to programming primitives.
- There are several more methods with varying complexity, but these are the most commonly used ones.

Summary

- The types of addressing that your hardware platform supports has a direct effect on the efficiency it can run algorithms written in high-level languages.
- This is an issue that *Instruction Set* architects must think about.
  - More on this next lecture!