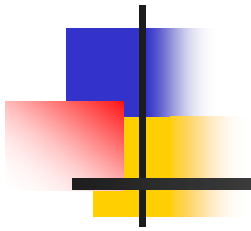
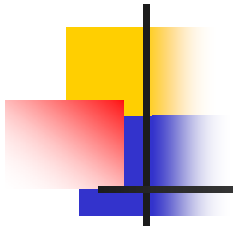


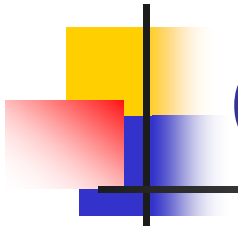
Operand Addressing And Instruction Representation

Alexandre David
1.2.05
adavid@cs.aau.dk



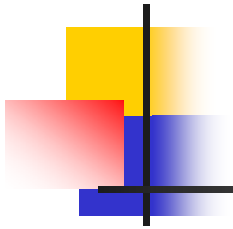
Introduction

- Previous chapter: processor types & instruction sets.
- This chapter:
  - how to represent instructions
  - how to specify operands



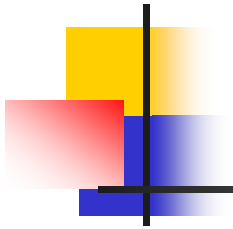
Operands Per Instruction

- Depends on the architecture
  - 0,1,2,3 (or more) address design
  - Few → easier to decode, need more elementary instructions to perform tasks, simpler, faster, smaller.
  - Many → [opposite]



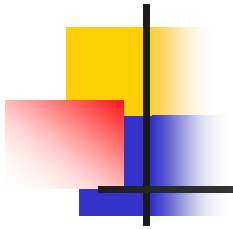
0-Address Design

- Operands are implicit.
- Typical for stack-based computers.
- Program:
  - push arguments
  - execute operators
    - consume arguments
    - produces result(s)
  - pop results
  - Ex: push X; push 7; add; pop X



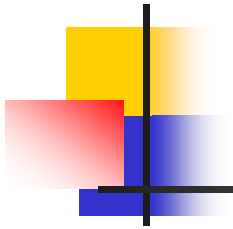
1-Address Architecture

- Similar to hand calculator.
- One explicit operand, one implicit operand (accumulator).
  - Accumulator = special register used for argument and result.
  - Ex: load X, add 7, store X



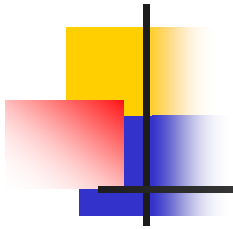
2 Operands Per Instruction

- 2 explicit operands: source & destination (used also as 2nd source).
- Good for memory copy.
- Ex: add 7,X



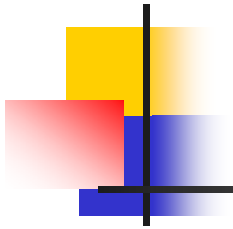
3 Operands Per Instruction

- 2 sources, 1 destination.
- Ex: add src1, src2, dst
  - add 7,X,X
  - add X,X,Y
  - add 0,X,Y



Operand Types

- Not all combinations are allowed in practice (efficiency, cost).
  - immediate value
  - register – value
  - register – address – memory reference
- Von Neumann Bottleneck
  - Bottleneck = memory. Operand addressing → access memory.
  - Justifies use of registers.
  - Memory accesses limit performance.



Operand Encoding

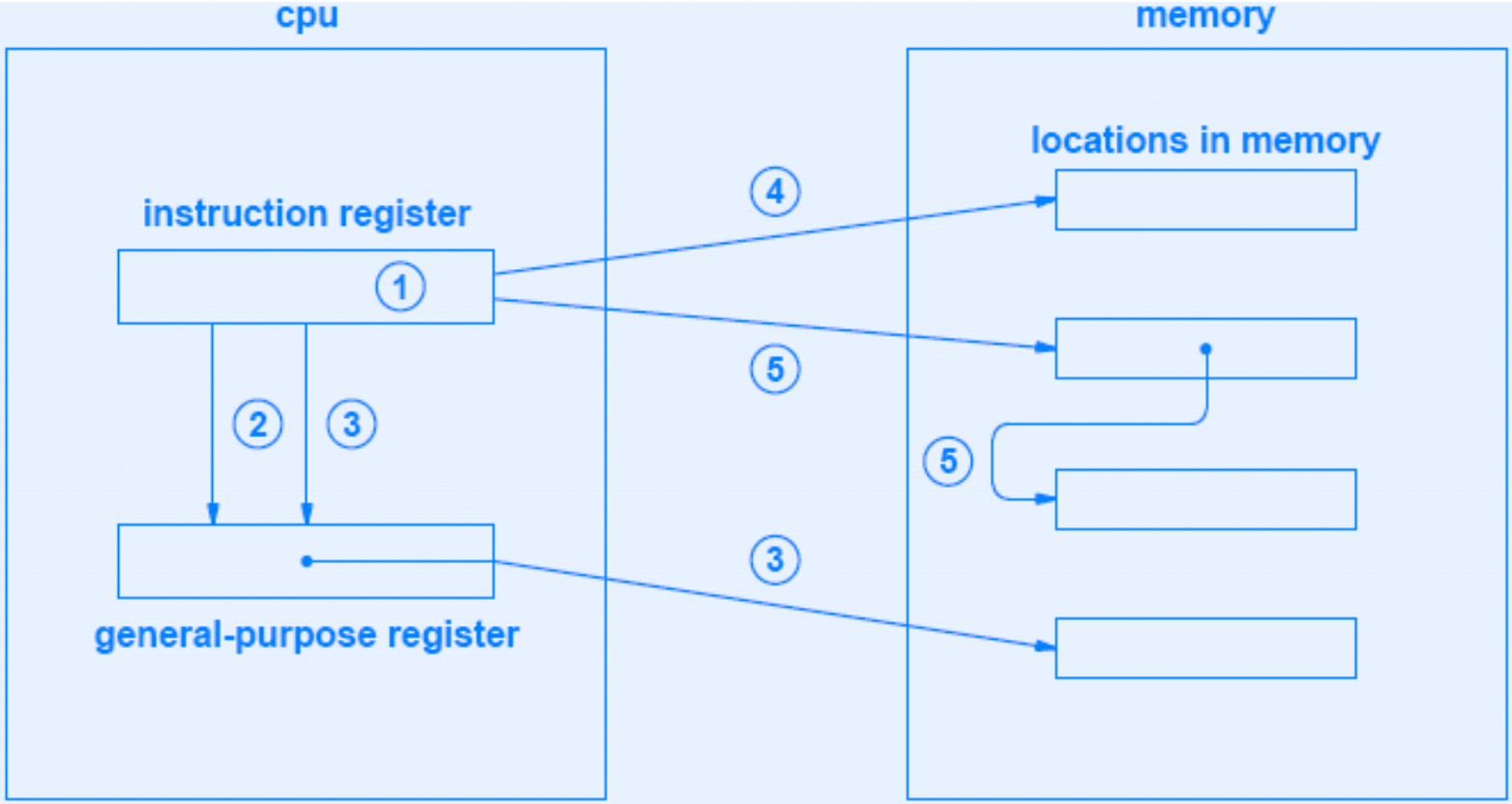
- Implicit operand encoding
  - opcode tells signature
  - more opcodes needed
- Explicit operand encoding
  - type in operand
  - more complex to decode

- See Intel’s instruction set manual.

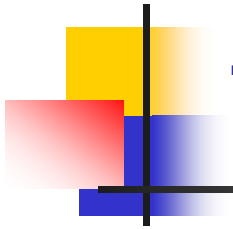
More Types of Operands

- Operands contain multiple items
  - typically register + offset
  - Intel example: (GNU asm)
    “mov 0x4(%%edi),%%eax”
    “lea (%%edi,%%ecx,$8),%%eax”
- Indirect reference (more expensive).

Types of Indirection

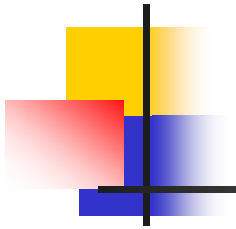


1 Immediate value (in the instruction)
2 Direct register reference
3 Indirect through a register
4 Direct memory reference
5 Indirect memory reference



Tradeoffs

- No perfect solution.
- Tradeoff between
  - ease of programming
  - fewer instruction
  - smaller instruction
  - range of immediate values
  - **faster fetch & decode**
  - **decreased hardware size**



Lessons

- Different kinds of operand encodings.
  - Fewer instructions means more complex types of encoding.
  - You should recognize how your high-level language is going to be executed.
  - x86 is indeed very complex.