William Stallings Computer Organization and Architecture

Chapter 9

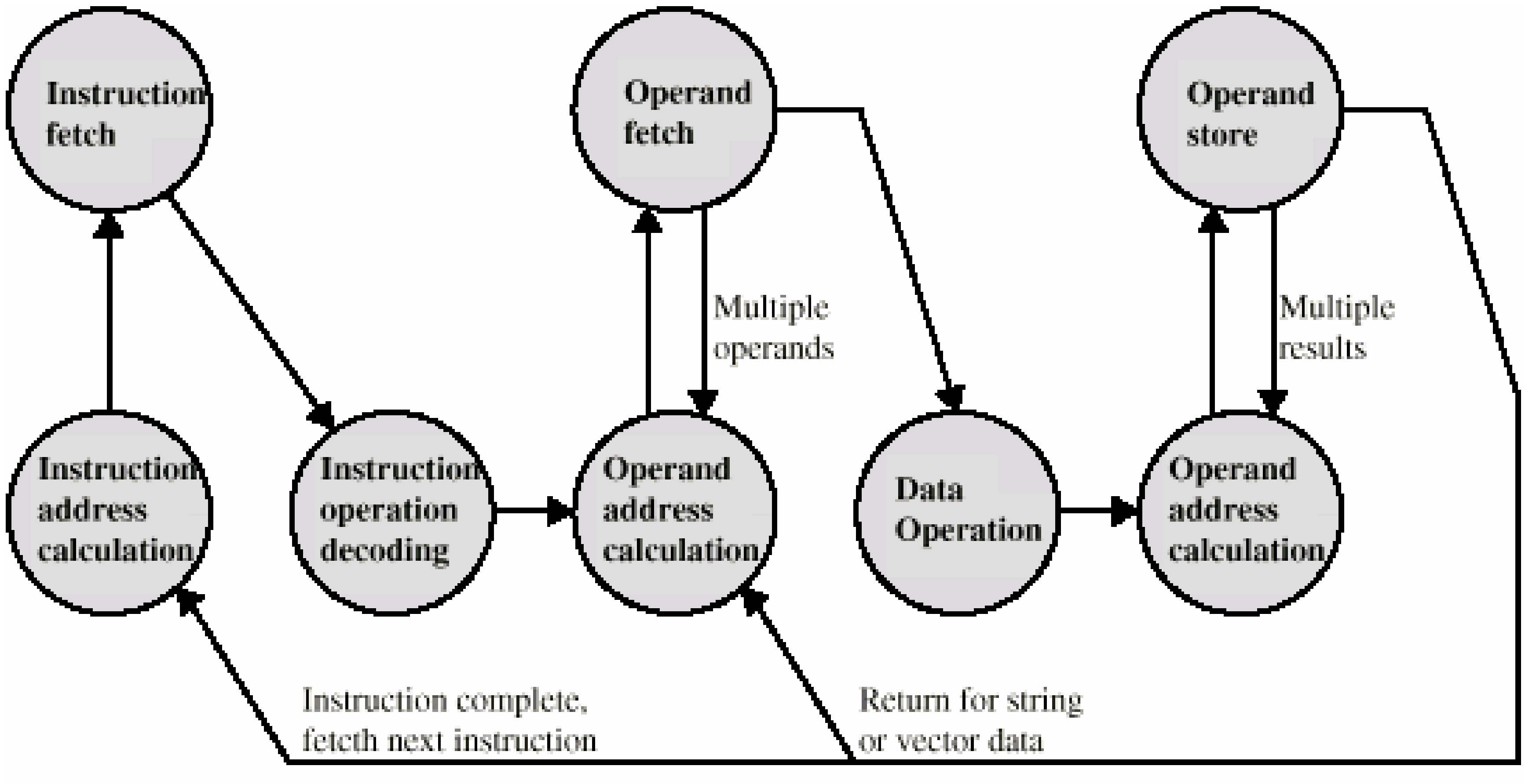
Instruction Sets:

Characteristics and Functions

What is an instruction set?

- The complete collection of instructions that are understood by a CPU
- The instruction set is the specification of the expected behaviour of the CPU
- How this behaviour is obtained is a matter of CPU implementation

Instruction Cycle



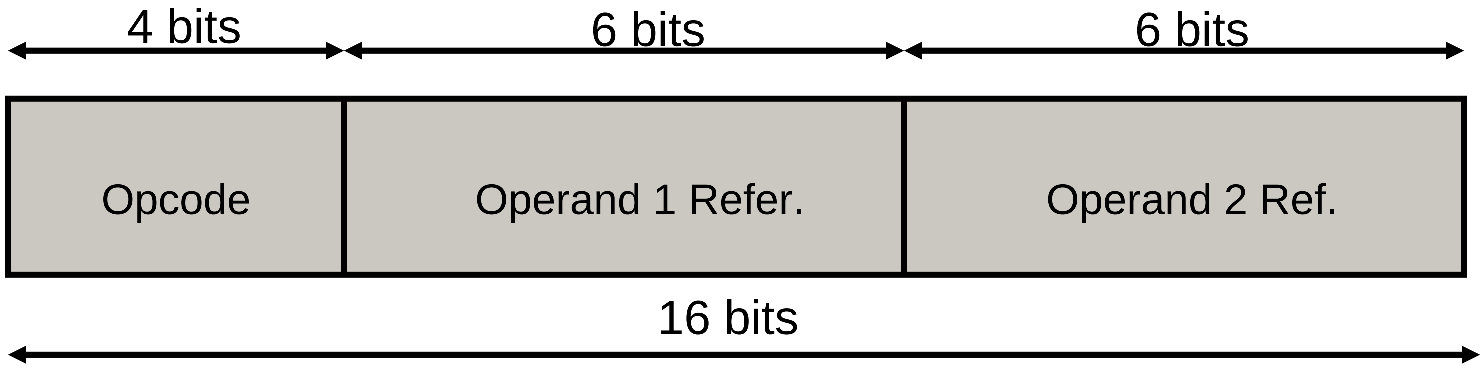
Elements of an Instruction

- Operation code (Opcode)
  - Do this
- Source Operand(s) reference(s)
  - To this (and this ...)
- Result Operand reference
  - Put the answer here
- The Opcode is the only mandatory element

Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (internal transfer and I/O)
- Program flow control

Instruction Representation



- There may be many instruction formats
- For human convenience a symbolic representation is used for both opcodes (MPY) and operand references (RA RB)
  - e.g. 0110 001000 001001 MPY RA RB
    (machine code) (symbolic - assembly code)

Design Decisions (1)

- Operation repertoire
  - How many opcodes?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length and structure of opcode field
  - Number and length of reference fields

Design Decisions (2)

- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later...)

Types of Operand references

- Main memory
- Virtual memory (usually slower)
- Cache (usually faster)

- I/O device (slower)
- CPU registers (faster)

Number of References/ Addresses/ Operands

- 3 references
  - ADD RA RB RC　　　RA+RB $\rightarrow$ RC
- 2 references (reuse of operands)
  - ADD RA RB　　　RA+RB $\rightarrow$ RA
- 1 reference (some implicit operands)
  - ADD RA　　　Acc+RA $\rightarrow$ Acc
- 0 references (all operands are implicit)
  - S_ADD　　　Acc+Top(Stack) $\rightarrow$ Acc

How Many References

- More references
  - More complex (powerful?) instructions
  - Fewer instructions per program
  - Slower instruction cycle
- Fewer references
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster instruction cycle

Example (1)

- Compute (A-B)/(A+(C*D)), assuming each of them is in a register which has cannot be modified. Additional registers X and Y can be used if needed. Try to minimize the number of operations
- 3 operands:

  <operation><destination><source-1><source-2>

  - MUL X C D   C*D -> X
  - ADD X A X   A+X -> X
  - SUB Y A B   A-B -> Y
  - DIV X Y X   Y/X -> X

Example (2)

- 2 operands (the destination is also the first source operand)

  <operation><destination><source>

  - MOV X C   C -> X
  - MUL X D   X*D -> X
  - ADD X A   X+A -> X
  - MOV Y A   A -> Y
  - SUB Y B   Y-B -> Y
  - DIV Y X   Y/X -> Y

Example (3)

- 1 operand (a given register, e.g. the accumulator, is both the destination and the first source operand)

  <operation><source>

  - LOAD C C -> Acc
  - MUL D Acc*D -> Acc
  - ADD A Acc+A -> Acc
  - STORE X Acc -> X
  - LOAD A A -> Acc
  - SUB B Acc-B -> Acc
  - DIV X Acc/X -> Acc

Example (4)

- 0 operands (all arithmetic operations make reference to pre-defined registers, e.g. the accumulator and the top of the stack, but moving value in and out accumulator and stack has 1 operand)
  - LOAD C — C -> Acc
  - PUSH D — D -> Top(Stack)
  - MUL — Acc*Top(Stack) -> Acc
  - PUSH Acc — Acc -> Top(Stack)
  - LOAD A — A -> Acc
  - ADD — Acc+Top(Stack) -> Acc
  - PUSH Acc — Acc -> Top(Stack)
  - PUSH B — B -> Top(Stack)
  - LOAD A — A -> Acc
  - SUB — Acc-Top(Stack) -> Acc
  - POP X — Top(Stack) -> X
  - DIV — Acc/Top(Stack) -> Acc

Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags
- (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

Instruction Types (more detail)

- Arithmetic
- Logical
- Conversion
- Transfer of data (internal)
- I/O
- Transfer of Control
- System Control

Arithmetic

- Add, Subtract, Multiply, Divide
- Signed Integer
- Floating point ?
- May include
  - Increment (a++)
  - Decrement (a--)
  - Negate (-a)

Logical

- Bit manipulation operations
  - shift, rotate, ...
- Boolean logic operations (bitwise)
  - AND, OR, NOT, ...
- Test operations
  - To set (indirectly through the ALU) control bits in the Program Status Word

Conversion

- e.g. Binary to Decimal

Transfer of data

- Specify
  - Source and Destination
  - Amount of data
- May be different instructions for different movements
  - e.g. MOVE, STORE, LOAD, PUSH
- Or one instruction and different addresses
  - e.g. MOVE B C, MOVE A M, MOVE M A, MOVE A S

Input/Output

- May be specific instructions
- May be done using data movement instructions (memory mapped)
- May be done by a separate controller (DMA)

Transfer of Control (1)

- Needed to
  - Take decisions (branch)
  - Execute repetitive operations (loop)
  - Structure programs (subroutines)
- Branch (examples)
  - BRA X: branch (i.e., go) to X (unconditional jump)
  - BRZ X: branch to X if accumulator value is 0

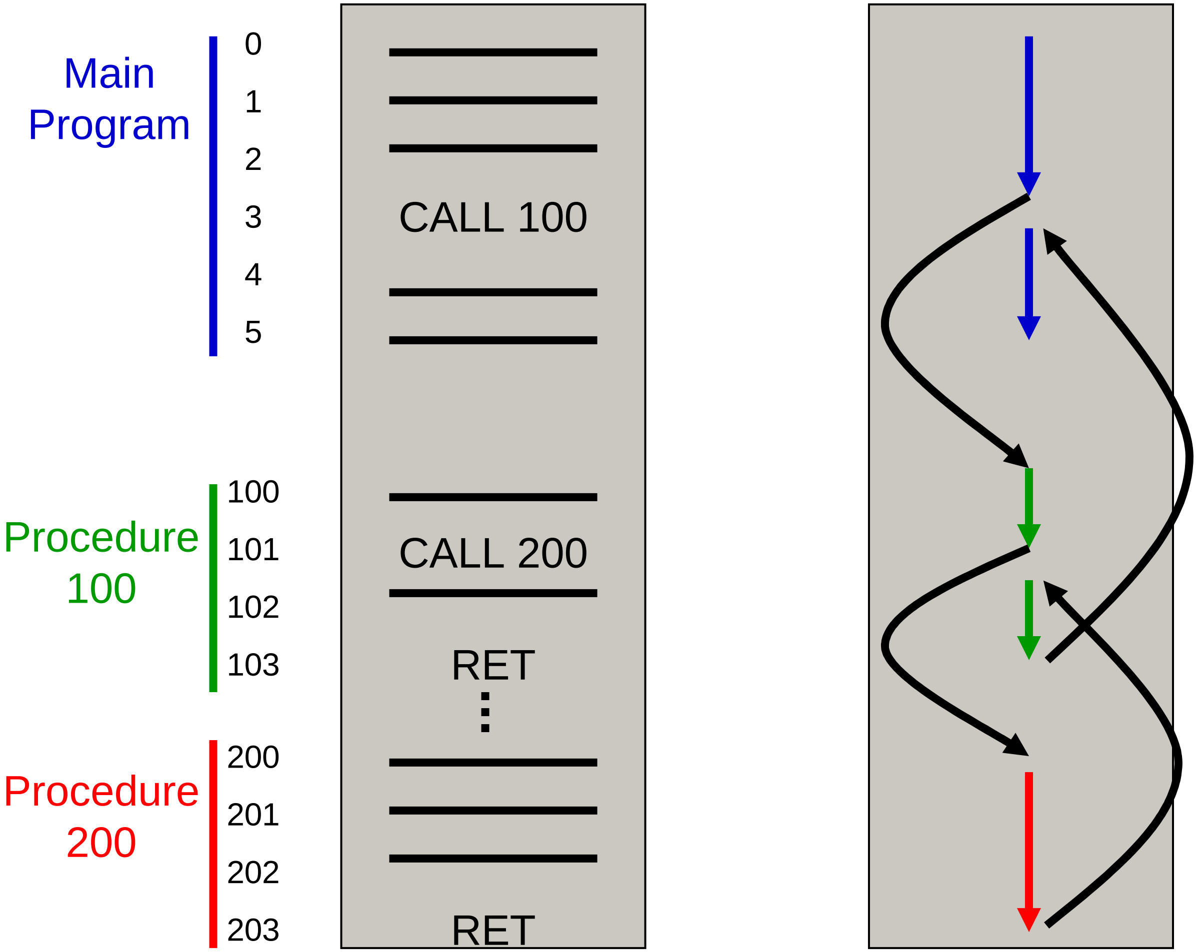
Transfer of control (2)

- Skip (example)
  - Increment register R and skip next instruction if result is 0

```
X:  ...
    ...
    ISZ R
    BRA X (loop)
    ...     (exit)
```

- Subroutine call (a kind of interrupt serving)

Subroutine (or procedure) call



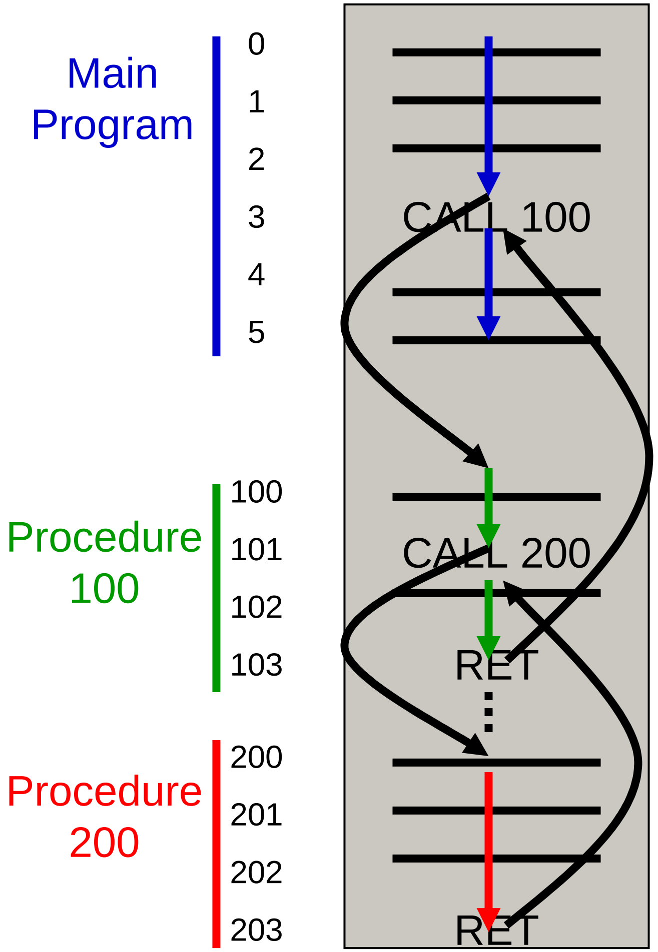
Alternative for storing the return address from a subroutine

- In a pre-specified register
  - Limit the number of nested calls since for each successive call a different register is needed
- In the first memory cell of the memory zone storing the called procedure
  - Does not allow recursive calls
- At the top of the stack (more flexible)

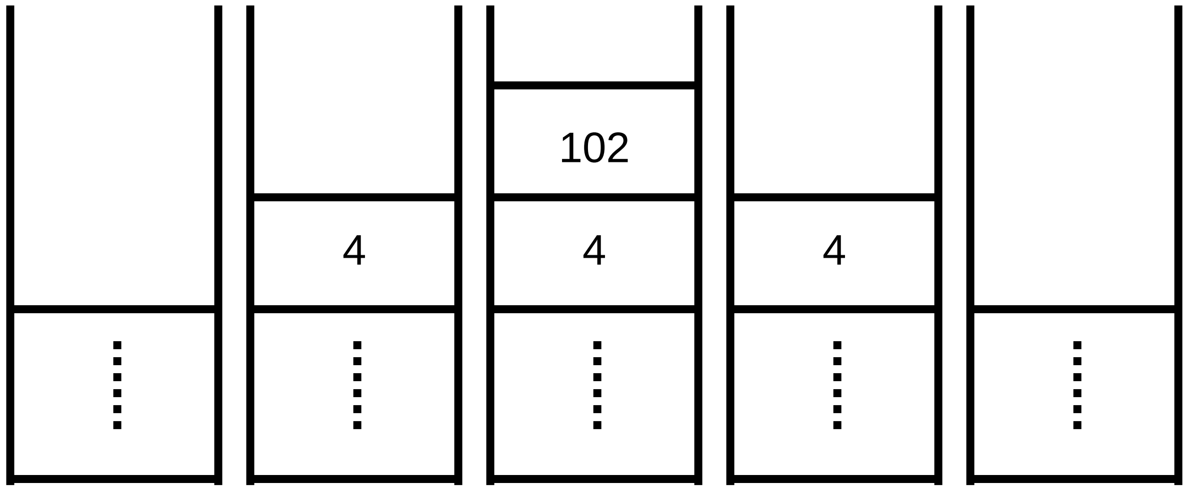
Return using the stack (1)

- Use a reserved zone of memory managed with a *stack* approach (last-in, first-out)
  - In a stack of dirty dishes the last to become dirty is the first to be cleaned
- Each time a subroutine is called, before starting it the return address is put on top of the stack
- Even in the case of multiple calls or recursive calls all return addresses keep their correct order

Return using the stack (2)



- The stack can be used also to pass parameters to the called procedure



Passing parameters to a procedure

- In general, parameters to a procedure might be passed
  - Using registers
    - Limit the number of parameters that can be passed, due to the limited number of registers in the CPU
    - Limit the number of nested calls, since each successive calls has to use a different set of registers
  - Using pre-defined zone of memory
    - Does not allow recursive calls
  - Through the stack (more flexible)

System Control

- For operating systems use it is convenient to have *reserved* instruction executable only by some operating system programs (e.g., to halt a running program).
- These privileged instructions may be executed only if CPU is in a specific state (or mode)
- *Kernel* or *supervisor* or *protected* mode

Byte Order

- What order do we read numbers that occupy more than one cell (byte)
- 12345678 can be stored in 4 locations of 8 bits each as follows

Address	Value (1)	Value(2)
184	12	78
185	34	56
186	56	34
186	78	12

- i.e. read top down or bottom up ?

Byte Order Names

- The problem is called Endian
- The system on the left has the least significant byte in the lowest address
- This is called *big-endian*
- The system on the right has the least significant byte in the highest address
- This is called *little-endian*

Standard...What Standard?

- Pentium (80x86), VAX are little-endian
- IBM 370, Motorola 680x0 (Mac), and most RISC are big-endian
- Internet is big-endian
  - Makes writing Internet programs on PC more awkward!
  - WinSock provides *htoi* and *itoh* (Host to Internet & Internet to Host) functions to convert