From Programs to Executions: An Odyssey in Language Translation

(with examples in Scheme)

Keith D. Cooper

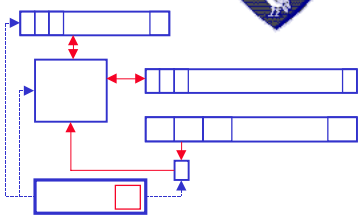
Rice University
Houston, Texas

December 2000

What commands does the "computer" run?

Computer's *instruction set*

- Low-level, imperative commands
  - Arithmetic operations
  - Memory operations
  - Control operations
  - Location-oriented programming
- We call these operations "assembly-language"



Arithmetic Operations
add r1, r2 => r3
sub r1, r2 => r3
mult r1, r2 => r3
div r1, r2 => r3

Memory Operations
load r1 => r2
store r1 => r2
loadi c1 => r2
copy r1 => r2

Control Operations
branch r1 -> r2
branchi r1 -> L2
call -> L1
return

Programming with Machine Operations

```
(define (summation n)
  (cond  [(= n 0) 0]
         [else  (+ n
                   (summation (sub1 n)))]))
```

This might become, after storage assignment & translation

```
100:  n
101:  result
```

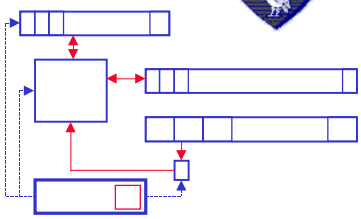
```
0:  loadi  1      ⇒ r0
1:  loadi  100    ⇒ r1
2:  eq?    r1 r1  ⇒ r11
3:  load   r1     ⇒ r2
4:  copy   r0     ⇒ r3
5:  add    r3 r2  ⇒ r3
6:  sub    r2 r0  ⇒ r2
7:  eq?    r2 r0  ⇒ r10
8:  branch r10    → 10
9:  branch r11    → 5
10: loadi  101    ⇒ r11
11: store  r2     ⇒ r11
12: stop
```

Assembly programming & assemblers in COMP/ELEC 320

Understanding How the Computer Works

One valuable tool is simulation

- Write a program that has the same behavior
- Models behavior of system
- Gives insight into its workings



Simulation is used in many ways

- Design of new systems
- Conduct experiments that are expensive or dangerous
- Train pilots in cases where loss is expensive

Simulating a computer shows us a lot about how it works

How Does the Computer Work?

```
;; an op is a structure
;;    (make-op fn arg1 arg2 arg3)
(define-struct op (fn arg1 arg2 arg3))

;; program memory holds 10k operations
(define Prog (build-vector 10240
                           (lambda (x) (make-op 'STOP 0 0 0)))

;; data memory holds 10k "word" of data
(define Data (build-vector 10240
                           (lambda (x) 0)))

;; the program counter holds the address of the current op
(define PC 0)
```


How Does the Control Unit Work?

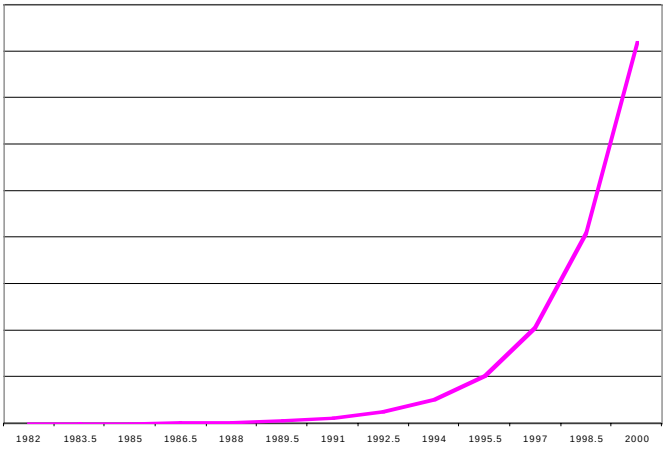
```
;; control-unit : void -> boolean
;; Purpose: execute the operations in program memory, in
;;          sequential order
(define (ControlUnit)
  (local [(define ThisOp (vector-ref Prog PC))]
    (begin
      (set! PC  (add1 PC))  ;; update next operation's position
      (cond [(symbol=? (op-fn ThisOp) 'STOP)  true]
            [else
              (begin
                (FunctionalUnit ThisOp)
                (ControlUnit)) ])
      )))
```

How Does the Functional Unit Work?

```
;; FunctionalUnit : op -> void
(define (FunctionalUnit AnOp)
  (local [ (define f   (op-fn AnOp))      (define r1 (op-arg1 AnOp))
           (define r2 (op-arg2 AnOp))   (define r3 (op-arg3 AnOp)) ]
    (cond
      [(symbol=? f 'ADD)
       (vector-set! Regs r3 (+ (vector-ref Regs r1) (vector-ref Regs r2)))]
      [(symbol=? f 'LOAD)
       (vector-set! Regs r2 (vector-ref Data (vector-ref Regs r1)))]
        • • •
      [(symbol=? f 'EQ?)
       (vector-set! Regs r3 (= (vector-ref Regs r1)(vector-ref Regs r2)))]
       [(symbol=? f 'BRANCH)
         (cond
             [(vector-ref Regs r1) (set! PC r2)] ;; not Regs[r2], but r2 itself
             [else void])]
  )))
```

Complete code on web site

Computers Keep Getting Faster



Processor power versus time, 1980 to 2000

Plot of
Moore's Law

1980-2000

How did this
happen?

They Are Running Faster

The clock frequency of processors has risen

- 1983: 10 MHz 68020 provided about 1 MIP ⇒ 10 cycles/operation
- 2000: 500 MHz PowerPC provides 500 MIPS + 1 GFLOP
- 2000: Pentiums in 700 to 800 MHz range ⇒ 1 cycle/operation

High-end chips are breaking the 1 GHz barrier

All this power has a downside, however

- Power consumption ∝ frequency^2
- Heat ∝ power
- Computation needs operands, needs memory

They Are Also Running More Operations

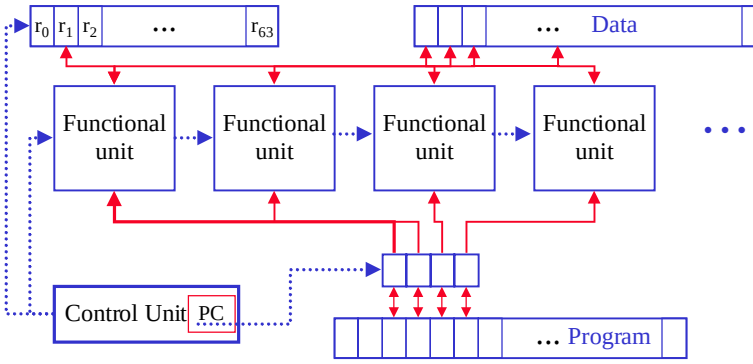
Programs contain parallelism

- Operations that can execute at the same time
- Can occur at the fine-grained level or on a larger scale

Computers can exploit parallelism

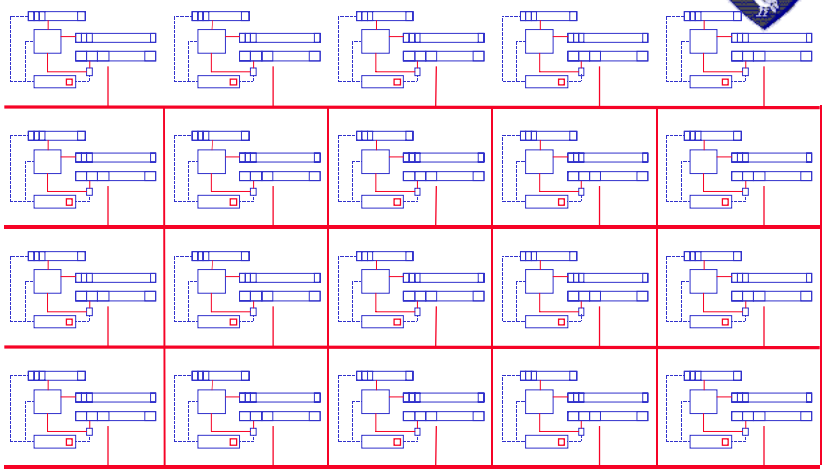
- Increase operations per cycle
- Use more hardware rather than faster hardware
- Many options
  - Single chip processor with many functional units
  - Custom-built machine with many individual nodes (computers)
  - Networks of workstations and/or PCs

Single Chip, Many Functional Units



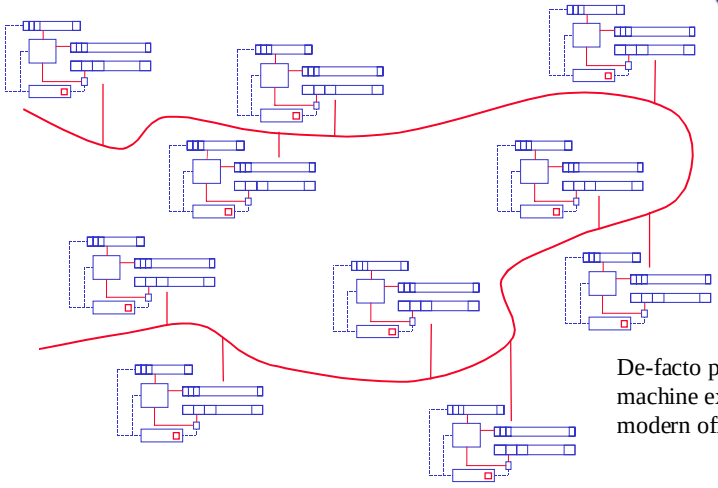
This takes advantage of instruction-level parallelism

Many Processors, Dedicated Interconnect



Multiprocessor Computer

Network of Workstations



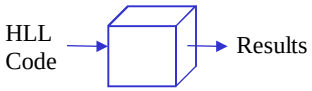
De-facto parallel machine exists in most modern offices !

Going From HLL to ASM

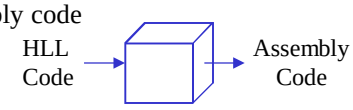
Assembly programming & assemblers in COMP/ELEC 320

How do we get from a high-level language to assembly code for some target machine?

- Write an interpreter in a working HLL
  - The base of DrScheme is written in C & C++
  - The rest is written in Scheme
  - Subject of COMP 311



- Write a compiler for the HLL
  - Translate HLL directly to assembly code
  - Must run the assembly code
  - Subject of COMP 412



Inside a Compiler

Chooses names, operations, maps *n*-ary operators into binary ops.

- Represent the program in some internal form

  (+ a b c) ⇒ (list + a b c)

- Traverse that data structure and generate code

(list + a b c) ⇒

```
add a,b  ⇒ t1
add t1,c ⇒ t2
print t 1
```

Along the way

- Do most of the things that the interpreter does!
- Plan, when compiler runs, how events will occur when the program actually runs
- Plan once, execute many times

Compiler versus Interpreter

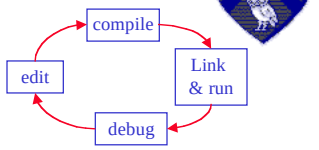
- The interpreter manages resources as demands arise
- The compiler must plan such use in advance
  - "Planning" involves designing and implementing techniques that can efficiently manage these issues
  - By working "off-line," issues like naming become easier
  - Most decisions can be made in small,constant amount of time

The Benefit

- Management & overhead time incurred once
- Management & overhead time reduced by moving off line

⇒ The running program is faster

The Complications of Using a Compiler



Requires the use of more tools

- Editor to enter program text
- Linker to tie compiled code together
- Debugger to relate post-explosion state back to source *(possibly)*

Environment is (typically) less standardized

- Depends on the libraries of code for "built-in" functionality
- Once we break the process up to allow compilation of parts, we can get into trouble with inconsistent or incomplete libraries
- The programmer now must keep track of these issues

Welcome to COMP 212 (and beyond)