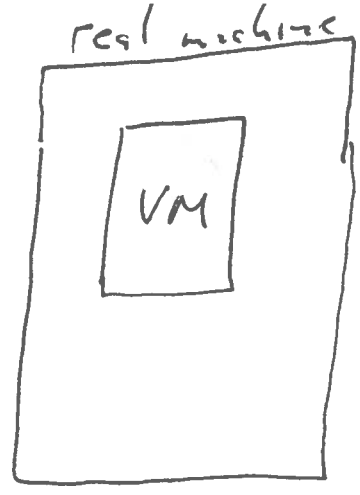
Representation of vm520 Programs

CS520

Dept. of Computer Science
Univ. of New Hampshire

Virtual Machines



system virtual machine —
emulate full operating system

process virtual machine —
only emulate one process

→ have a long history in Computer Science

Pcode
Forth

Python
Lua
Java
C#

VM 520

toy VM for executing 32-bit integer and floating-point computations

multiprocessor VM

1MB memory (20-bit addresses) of 32-bit words

each processor has a set of registers:

- 13 data registers: r0, r1, ..., r12
- frame pointer - Fp
- stack pointer - SP
- program counter - PC

all registers are 32 bits

Fetch/Execute Cycle

1. Fetch instruction at address in pc.
2. Add one to the pc.
3. Execute the instruction.
4. Go to step 1.

VM520 Program Representation

all instructions are 32-bits

8 different instruction formats

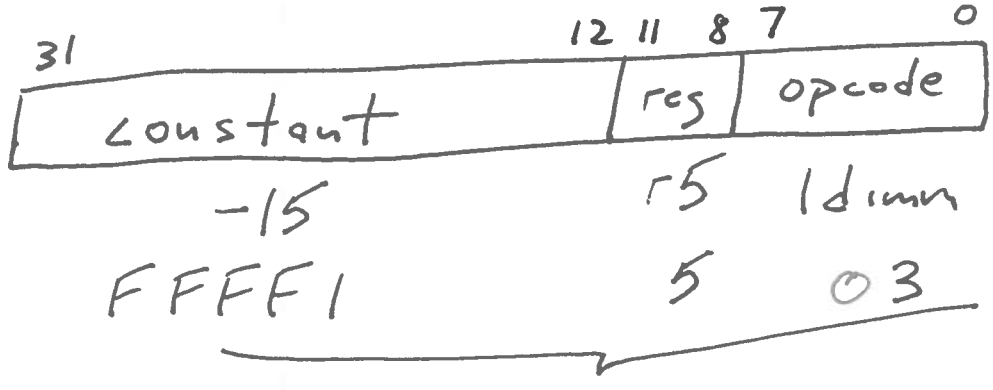
immediate mode constants & offsets

↳ avail.ble immediately in the instruction

two's complement form

PC-relative addresses

ldimm r5, -15



$15 = 0000\ 1111_2$

$-15 \quad 1111\ 0000$

$+1$

$1\_\_ \quad 1111\ 0001$

So Cool

Programs as Data!

sumVector.asm

```
#
# This vm520 program simply sums together a list of numbers.
#
# The label "sum" is exported to allow the main program invoking
# the virtual machine to retrieve the answer.
#
# The labels "top" and "done" are exported to allow tests of the disassembler.
#

export sum
export top
export done

# execution will start here (at address 0)
jmp skipData
sum:
  word 0
len:
  word 5
vector:
  word 1
  word 2
  word 3
  word 4
  word 5
skipData:
  ldimm  r0, 0          # r0 is the loop index
  load   r1, len        # r1 is the upperbound for the loop
  ldaddr r2, vector     # r2 is a pointer to an vector element
  ldimm  r3, 0          # r3 is the running sum
  ldimm  r4, 1          # r4 always contains 1, the loop increment
top:
  beq    r0, r1, done   # loop exit condition
  ldind  r5, 0(r2)      # fetch vector[i]
  addi   r3, r5         # add it to the sum
  addi   r2, r4         # increment pointer
  addi   r0, r4         # increment loop index
  jmp    top
done:
  store  r3, sum
  halt
```

Assembly Language

PC initialized to 0.

execution starts here

label

r0	r1	r2	r3	r4	r5
0	5	addr (vector)	0	1	1
1		addr +1	1		2
2		addr +2	3		3
3		addr +3	6		4
4		addr +4	10		5
5		addr +5	15		