Representation of vm520 Programs

CS 520
Dept. of Computer Science
Univ. of New Hampshire
Virtual Machines

- System virtual machine — emulate full operating system
  - only emulate one process
  - have a long history in Computer Science

  - Pcode
  - Python
  - Forth
  - Lua
  - Java
  - C#
VM 520

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

multi-processor 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.
VM 520 Program Representation

all instructions are 32-bits

8 different instruction formats

immediate mode constants & offsets

- available immediately in the instruction
two's complement form

PC-relative addresses
1dimm 15  -15

\[
\begin{array}{cccccc}
31 & 12 & 11 & 8 & 7 & 0 \\
\text{constant} & \text{reg} & \text{opcode} \\
-15 & 15 & 1dimm \\
\end{array}
\]

FFFF1  5  03

15 = 0000 1111_2

-15  1111 0000 +1

FFFF1503

So Cool
Program as
Data!
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.

```assembly
export sum
export top
export done

# execution will start here (at address 0)
jmp skipData

sum:
    word 0 15
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
    ldimd 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
```

PC int. Load to φ.