Program Translation

- Suppose we want to execute the following statement in a *high-level programming language* (e.g.,C):
  - a = b + c;
- The C compiler is going to take that statement, and translate it into *assembly language* for a particular CPU architecture:

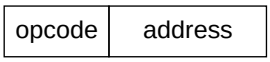
```
LOAD    20    ; get b (stored at 20)
ADD     21    ; add c (stored at 21)
STORE   22    ; store in a (at 22)
```

- The assembler will translate those assembly language statements into *machine language*:

```
100 10100
000 10101
101 10110
```


1-Operand (Accumulator) Instruction Format

- Instruction format:

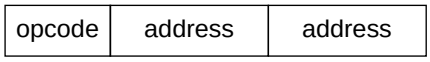


- Uses an *accumulator* to store the "current" value
  - Can directly access only a single operand
  - Arithmetic instructions
    - use the accumulator as the first operand
    - store the result in the accumulator
- Instructions: (*addr* = address)

ADD *addr*	ACC = ACC + M[*addr*]
SUB *addr*	ACC = ACC – M[*addr*]
MPY *addr*	ACC = ACC • M[*addr*]
DIV *addr*	ACC = ACC / M[*addr*]
LOAD *addr*	ACC = M[*addr*]
STORE *addr*	M[*addr*] = ACC

2-Operand Instruction Format

- Instruction format:

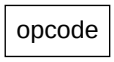


- Instructions are in form: *op dest, src*
  - Can directly access two operands
- Instructions: (*src* = source, *dst* = destination, both refer to memory addresses)

ADD *dst*, *src*	M[*dst*] = M[*dst*] + M[*src*]
SUB *dst*, *src*	M[*dst*] = M[*dst*] – M[*src*]
MPY *dst*, *src*	M[*dst*] = M[*dst*] • M[*src*]
DIV *dst*, *src*	M[*dst*] = M[*dst*] / M[*src*]
MOVE *dst*, *src*	M[*dst*] = M[*src*]

0-Operand (Stack) Instruction Format

- Instruction format:



- Uses a *stack* to store operands
  - Values are entered by reading them from memory and *pushing* them onto the stack
  - Arithmetic instructions operate on the "top" 1 or 2 operands on the stack, replacing those operands with the result
  - Values are stored by *popping* the stack — removing the value from the top of the stack and storing it in memory
- Instructions: (TOS = top of stack)

PUSH *addr*	TOS = M[*addr*]
POP *addr*	M[*addr*] = TOS
ADD	TOS = TOS-1 + TOS
SUB, MPY, DIV	…

Instruction Formats

- Example: M[102] = M[100] – M[101]
- 1-Operand (Accumulator) Format

```
LOAD    100
SUB     101
STORE   102
```

- 2- Operand Format

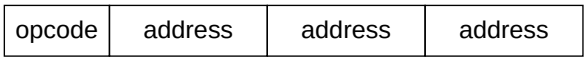
```
MOVE    105, 100        MOVE 102, 100
SUB     105, 101        SUB  102, 101
MOVE    102, 105
```

- 0- Operand (Stack) Format

```
PUSH    100
PUSH    101
SUB
POP     102
```


3-Operand Instruction Format

- Instruction format:



- Instructions are in form:

op dst, src1, *src2*

  - Can directly access <u>three</u> operands

- Instructions: (*src1* = source 1, *src2* = source 2, *dst* = destination, all three refer to memory addresses)

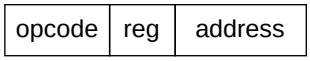
ADD *dst*, *src1*, *src2*	M[*dst*] = M[*src1*] + M[*src2*]
SUB *dst*, *src1*, *src2*	M[*dst*] = M[*src1*] – M[*src2*]
MPY *dst*, *src1*, *src2*	M[*dst*] = M[*src1*] • M[*src2*]
DIV *dst*, *src1*, *src2*	M[*dst*] = M[*src1*] / M[*src2*]

Memory vs. Registers

- (Off-chip) main memory
  - Very big
  - Slow
- (On-chip) CPU registers
  - Small number available
  - Very fast
- For the 1-address, 2-address, and 3-address operand formats that we've shown, any of the addresses can generally be replaced by a register
  - 1 address: ADD R2
  - 2 address: ADD R5, 100
  - 3 address: ADD 101, R3, R4
- How can we take advantage of registers?

RISC LOAD/STORE Instruction Format

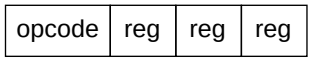
- LOAD and STORE instructions are the only ones that can access memory:



LOAD R*dst*, *addr*

STORE *addr*, R*src*

- Other instructions must operate solely on registers:



ADD R*dst*, R*src1*, R*src2*

SUB R*dst*, R*src1*, R*src2*

MPY R*dst*, R*src1*, R*src2*

DIV R*dst*, R*src1*, R*src2*

The RISC / CISC Difference

- CISC — *Complex* Instruction Set Computer (example: Intel Pentium)
  - Many instructions and addressing modes, may perform complicated operations
  - Many different architectures
- RISC — *Reduced* Instruction Set Computer (example: Sun SPARC, HP PA-RISC, Motorola PowerPC)
  - Few instructions and addressing modes, perform simple, well-defined operations
    - LOAD / STORE architecture
      - Only LOAD and STORE instructions access the main memory
      - Other instructions operate solely on registers
  - Other characteristics of RISC machines
    - Single-cycle execution of instructions
    - High degree of pipelining
    - Overlapping register windows

Worksheet

- Write assembly language code for the following two statements, in the 1-, 2-, and 0-operand instruction formats, assuming that

  variable *a* is stored at address 100,
  b is at 101,
  c is at 102,
  d is at 103,

  and addresses 105–109 can be used for temporary storage of intermediate results.

  - a = b + c*d
  - a = b – (c/d)

Homework #3 — Due 10/12/98 (Part 3)

4. Write assembly language code for the statement “a = (d+c) – a”, in the 0-, 1-, and 2-operand instruction formats, assuming:

   variable *a* is stored at address 20,
   b is at 21, *c* is at 22, *d* is at 23,
   and addresses 25–29 can be used for temporary storage of intermediate results.

   Do not destroy the values in variables b, c, or d.

(This is the last question on Homework #3)