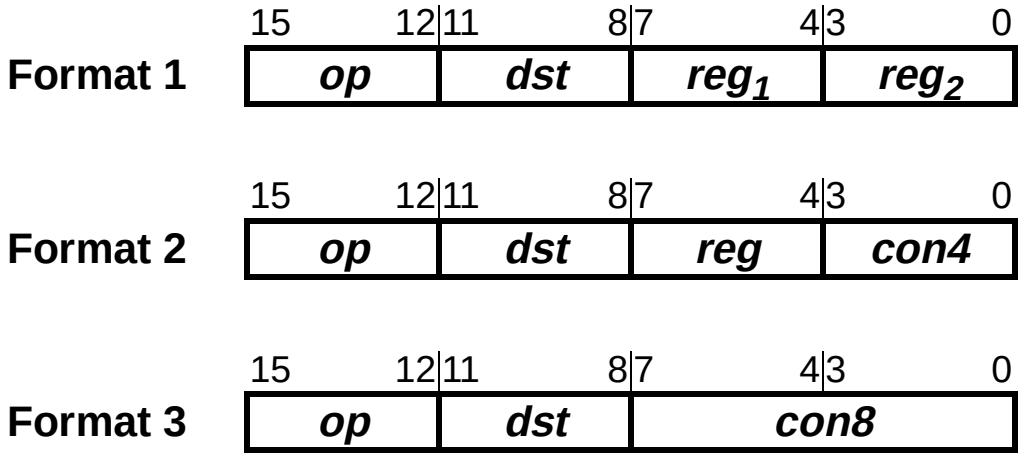
# Lecture 8. TOY Instructions

- A program is a sequence of instructions
- An instruction is a 16-bit word, interpreted in one of many possible ways
- 3 instruction 'formats,' 16 different instructions




***Format 1***

| | | | |
|---|---|---|---|
| 0 | halt | C | xor |
| 1 | add | D | and |
| 2 | subtract | E | shift right |
| 3 | multiply | F | shift left |

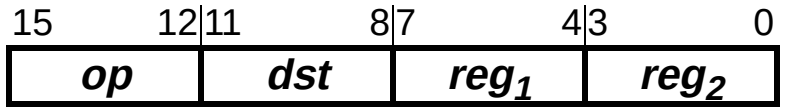
***Format 2***

| | |
|---|---|
| 9 | load |
| A | store |

***Format 3***

| | |
|---|---|
| 4 | system call |
| 5 | jump |
| 6 | jump if less |
| 7 | jump indirect |
| 8 | jump and link |
| B | load immediate |

# Format 1 Instructions

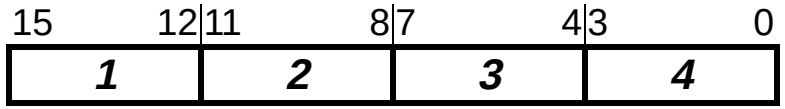



- **Format 1 instructions are *register-to-register* instructions**

  **Interpret *dst*, $reg_1$, and $reg_2$ as register numbers**

  **Take operands from $reg_1$ and $reg_2$, and put the result in *dst***

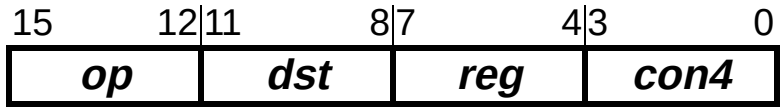
  **Example: $1234_{16}$ means $R_2 \leftarrow R_3 + R_4$**




**Stores the sum of the contents of registers $R_3$ and $R_4$ into register $R_2$**

| | | |
|---|---|---|
| **$2116_{16}$** | **$R_1 \leftarrow R_1 - R_6$** | |
| **3267** | **$R_2 \leftarrow R_6 \times R_7$** | |
| **C512** | **$R_5 \leftarrow R_1$ ^ $R_2$** | **exclusive OR** |
| **D645** | **$R_6 \leftarrow R_4$ & $R_5$** | **logical AND** |
| **E056** | **$R_0 \leftarrow R_5$ >> $R_6$** | **shift right** |
| **F764** | **$R_7 \leftarrow R_6$ << $R_4$** | **shift left** |
| **0000** | | **halt** |

# Format 2 Instructions




- **Format 2 instructions are *memory operation* instructions**

  **Interpret *dst* and *reg* as register numbers, *con4* as a 4-bit *unsigned* constant**

  **Compute the *effective address* *reg* + *con4***

- ***Load* copies a word *from memory* at the effective address *to register dst***

  **$9123_{16}$ means $R_1 \leftarrow M[R_2 + 3]$**

  **Copy the contents of the memory location specifed by adding 3 to the contents of register $R_2$ to register $R_1$**

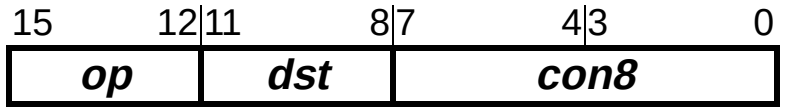
- ***Store* copies a word *from register dst to memory* at the effective address**

  **$A765_{16}$ means $M[R_6 + 5] \leftarrow R_7$**

  **Copy the contents of register $R_7$ to the memory location specifed by adding 5 to the contents of register $R_6$**

- **When *con4* is 0, load/store are sometimes called *indirect* load/store**

# Format 3 Instructions




- **Most of the format 3 instructions are *control* instructions**

  **Interpret *dst* as a register number, *con8* as an 8-bit *unsigned* constant or address**

  **Compute a *jump address* as either *con8* or *dst***

  **Set PC to that address**

  **Oddballs: system call (4) and load immediate (B)**

- ***Load immediate* copies *con8* to register *dst***

  **B234$_{16}$ means $R_2 \leftarrow 34_{16}$ set register R2 to $34_{16}$**

  **Use load immediate to copy the contents of a register to another register:**

  | | | |
  |---|---|---|
  | **B000$_{16}$** | $R_0 \leftarrow 0$ | set $R_0$ to 0 |
  | **1320** | $R_3 \leftarrow R_2 + R_0$ | set $R_3$ to $R_2 + R_0 = R_2 + 0 = R_2$ |

- ***System call* invokes actions that need special permission, like I/O**

  ***con8* specifies the system call 'action code', *dst* may specify an operand**

  **4402$_{16}$ writes the contents of $R_4$ to the standard output**

# Jump Instructions

- ***Jump*** **instructions change the PC to *con8*, or to the contents of *dst***

  **jump**

  **`5062`$_{16}$ PC $\leftarrow 62_{16}$**

  **The next instruction will be taken from M[$62_{16}$]**

  **jump if less**

  **`6362` PC $\leftarrow 62_{16}$ *if* the contents of $R_3 < 0$**

  **jump indirect**

  **`7500` PC $\leftarrow R_5$**

  **The next instruction will be taken from the address in $R_5$**

  **jump and link**

  **$3A_{16}$ `8462` $R_4 \leftarrow$ PC, PC $\leftarrow 62_{16}$**
  **3B**

  **The contents of the PC ($3B_{16}$) are saved in $R_4$, then the PC is set to $62_{16}$**
  **The next instruction will be taken from M[$62_{16}$]**

  **Used for *function linkage* — calls and returns**

- **All instructions of format 3 use a constant as one operand and a register or the program counter as the other operand.**

# Example: Bit Twiddling

- Set $b_0$ of $R_4$ to $b_{10}$ ^ $b_3$ from $R_1$, clear $b_1$–$b_{15}$ in $R_4$

```
R4 = ((R1>>10) ^ (R1>>3)) & 1;
```

```
1010 0111 0111 0010    R1
0000 0000 0010 1001    R1>>10
0001 0100 1110 1110    R1>>3
0001 0100 1100 0111    (R1>>10) ^ (R1>>3)
0000 0000 0000 0001    ((R1>>10) ^ (R1>>3)) & 1
```

Assuming $R_1$ is initialized to $A772_{16}$

```
00: B000    R0 <- 00
01: 1210    R2 <- R1 + R0 = A772
02: 1310    R3 <- R1 + R0 = A772
03: B50A    R5 <- 0A
04: B603    R6 <- 03
05: E225    R2 <- R2 >> R5 = 0029
06: E336    R3 <- R3 >> R6 = 14EE
07: C323    R3 <- R2 ^ R3 = 14C7
08: B401    R4 <- 01
09: D443    R4 <- R4 & R3 = 0001
```

# Example: Polynomial Evaluation

- **Evaluate $ax^2 + bx + c = 2x^2 + 3x + 9$ at $x = 10$ ($239_{10} = EF_{16}$)**

  **Store the 'data' in locations $30–33_{16}$**

```
30: 000A    x
31: 0002    a
32: 0003    b
33: 0009    c
```

- **Use Horner's method: rewrite $ax^2 + bx + c$ as $(ax + b)x + c$**

```
10: B330    R3 <- 30
11: 9430    R4 <- M[R3+00] = M[30] = 000A          x
12: 9531    R5 <- M[R3+01] = M[31] = 0002          a
13: 3554    R5 <- R5 * R4 = 0014                   a×x
14: 9632    R6 <- M[R3+02] = M[32] = 0003          b
15: 1556    R5 <- R5 + R6 = 0017                   a×x + b
16: 3554    R5 <- R5 * R4 = 00E6                   (a×x + b)×x
17: 9633    R6 <- M[R3+03] = M[33] = 0009          c
18: 1556    R5 <- R5 + R6 = 00EF                   (a×x + b)×x + c
19: 4502    system call 2: print R5 = 00EF
1A: 0000    HALT
```

- **Polynomial evaluation for arbitrary x**

  **many applications, one *raison d'etre* for early computers**