CHAPTER XIII

INSTRUCTION SET ARCHITECTURE (ISA)

READ INSTRUCTIONS FREE-DOC ON COURSE WEBPAGE

ISA
INTRODUCTION

- We have now considered the beginnings of the internal architecture of a computer.
  - With this, we considered microcode operations for performing simple data routing and calculations in one clock cycle.
- As a programmer, we don’t want to interface with the microprocessor and manually send each and every control signal as is done with microcode.
  - We would prefer to abstract the instruction sent to the microprocessor.
  - Let the microprocessor designer handle the decoding of the abstracted instruction into the microcode control operations.
- Start to define an assembly langauge! MIPS R3000/4000!

PROGRAM PATH

TRANSLATING CODE

- Below is the process for translating a program to machine opcodes.

High level program
e.g. C, C++, Pascal, Java

Compiler translates program

```
add $10, $8, $9
xor $13, $11, $12
lw $15,0($16)
```

Assembly language program

Assembler converts to machine code

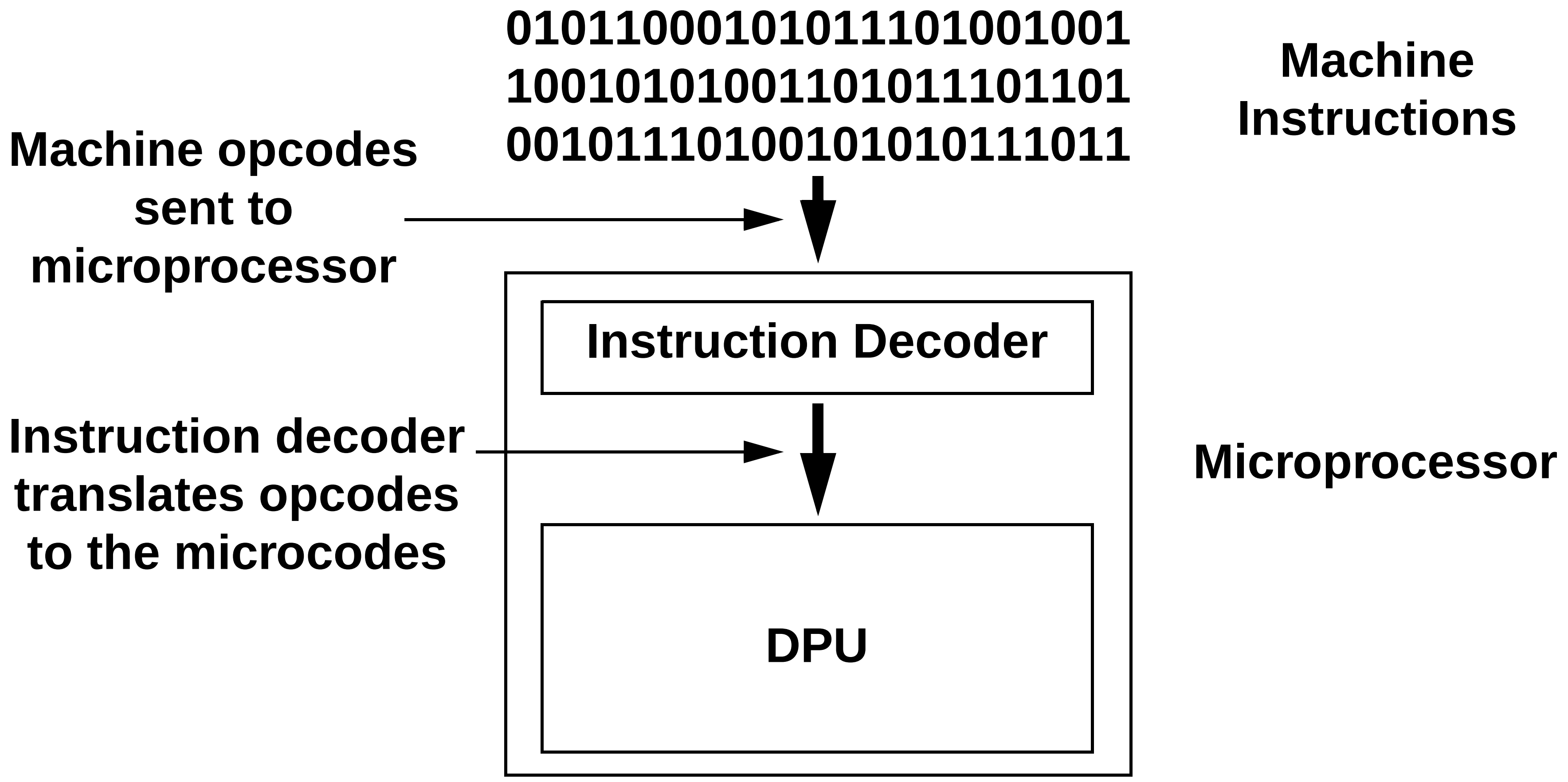
```
010110001010111010010001
100101010011010111011 01
001011101001010101110 11
```

Machine instructions

PROGRAM PATH

EXECUTING CODE

- Once the opcodes are given to the microprocessor, it translates the opcode instructions to the microcodes operations we discussed.



MIPS ASSEMBLY

MIPS REGISTER NAMES

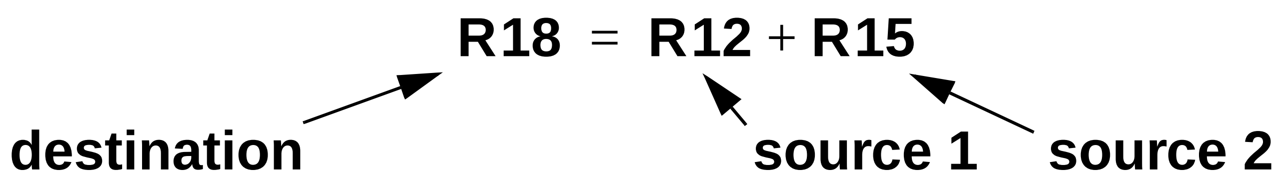
- For MIPS assembly, many registers have alternate names or specific uses.

Register	Name(s)	Use
0	**$zero**	**always zero (0x00000000)**
1		**reserved for assembler**
2-3	**$v0-$v1**	**results and expression evaluation**
4-7	**$a0-$a3**	**arguments**
8-15	**$t0-$t7**	**temporary values**
16-23	**$s0-$s7**	**saved values**
24-25	**$t8-$t9**	**temporary values**
26-27		**reserved for operating system**
28	**$gp**	**global pointer**
29	**$sp**	**stack pointer**
30	**$fp**	**frame pointer**
31	**$ra**	**return address**

MIPS ASSEMBLY

BASIC INST. FORMAT

- Need to consider an assembly language example. We will use the MIPS R3000/4000 assembly so that you can refer to the Instruction free-doc.
- MIPS R3000/4000 assembly instruction format:
  - The *majority* of MIPS instructions have the following assembly language instruction format.
    - **<inst mnemonic> <destination>, <source 1>, <source 2>**
  - You can see that this instruction format fits the register transfer level notation discussed with the single cycle DPU



MIPS ASSEMBLY

REGISTER FORMAT INST.

- Register format (R-format) instructions
  - Many MIPS instructions have the following format for register to register type binary operations.
    - **<instr> $<write register>, $<read register 1>, $<read register 2>**
  - An example of this is
    - **add $10, $8, $9**
  - This is the same as with our register transfer level operation
    - **R10 = R8 + R9**

MIPS ASSEMBLY

REGISTER INSTRUCTIONS

- Below is the basic list of register format MIPS instructions.

Instruction	Interpretation
add $10, $8, $9	**R10 = R8 + R9**
sub $10, $8, $9	**R10 = R8 - R9**
and $10, $8, $9	**R10 = R8 and R9**
or $10, $8, $9	**R10 = R8 or R9**
xor $10, $8, $9	**R10 = R8 xor R9**
sa $10, $8, $9 (shift arithmetic)	**Shift R8 by R9 and store in R10**
sl $10, $8, $9 (shift logical)	**Shift R8 by R9 and store in R10**
rot $10, $8, $9 (rotate)	**Rotate R8 by R9 and store in R10**
lw $10, 0($8)	**R10 = M[0+R8]**
sw $10, 0($8)	**M[0+R8] = R10**

MIPS ASSEMBLY

IMMEDIATE INST. FORMAT

- Immediate format (I-format) instructions
  - Many MIPS instructions have the following format for register to register type binary operations.
    - **<instr> $<write register>, $<read register>, <immediate value>**

Note: No $ for last argument

  - An example of this is
    - **addi $10, $8, 4**

Again, no $ for immediate value

Note: Include “i” to indicate an immediate value is used.

  - This is the same as with our register transfer level operation
    - **R10 = R8 + 4**

MIPS ASSEMBLY

IMMEDIATE INSTRUCTIONS

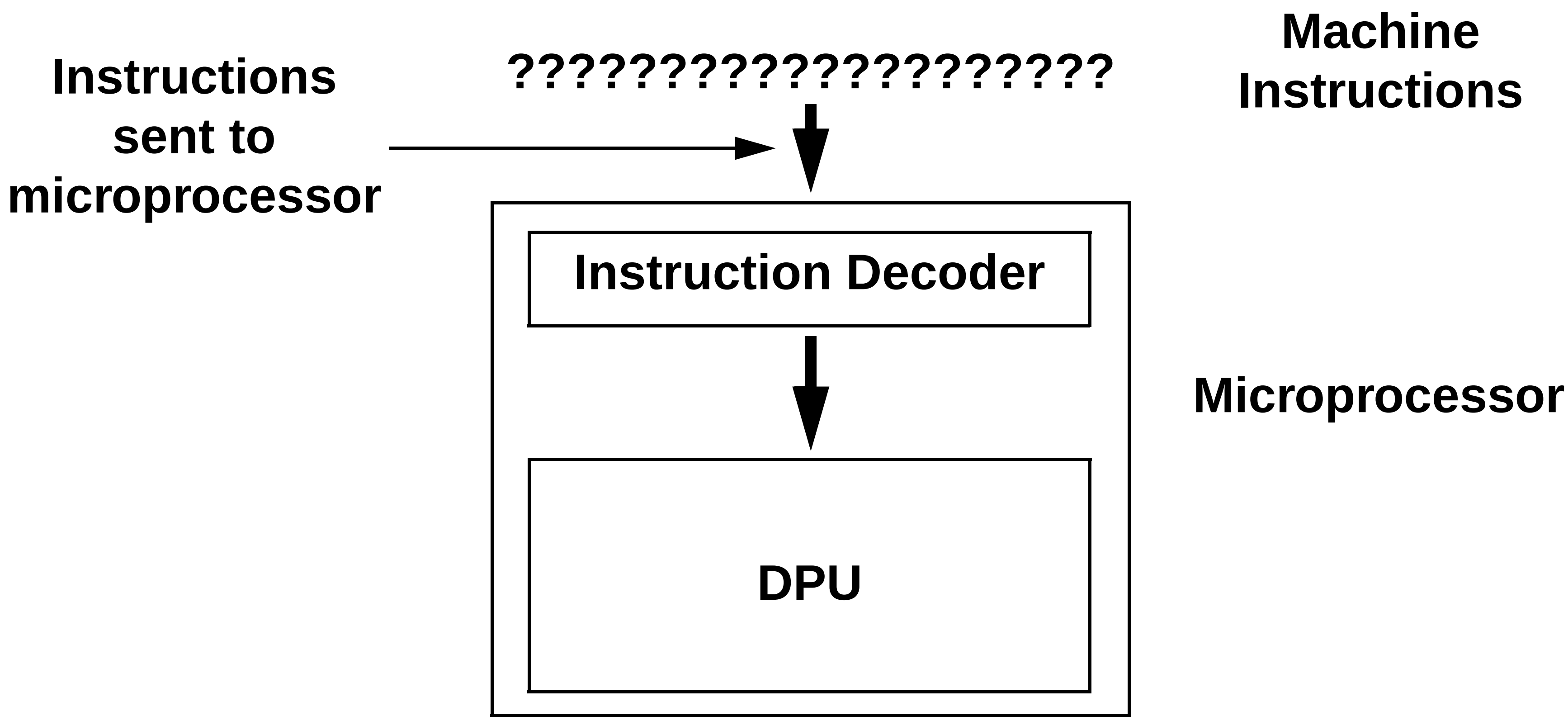
- Below is the basic list of immediate format MIPS instructions.

Instruction	Interpretation
addi $10, $8, 4	**R10 = R8 + 4**
subi $10, $8, 4	**R10 = R8 - 4**
andi $10, $8, 4	**R10 = R8 and 4**
ori $10, $8, 4	**R10 = R8 or 4**
xori $10, $8, 4	**R10 = R8 xor 4**
sai $10, $8, 4 (shift arithmetic)	**Shift R8 by 4 and store in R10**
sli $10, $8, 4 (shift logical)	**Shift R8 by 4 and store in R10**
roti $10, $8, 4 (rotate)	**Rotate R8 by 4 and store in R10**
lw $10, 4($0)	**R10 = M[4+R0]**
sw $10, 4($0)	**M[4+R0] = R10**

INST. SET ARCH.

INSTRUCTION FORMATS

- How should the assembly be translated to machine code?



- Have to consider what control signals the DPU requires!
- How do we abstract from the DPU's requirements?

INST. SET ARCH.

OPCODES

- First important part of a machine instruction is known as the operational codes (opcodes).
  - An **opcode** indicates what **major operation** to perform.
    - Example major operations:

      add, subtract, AND, OR, NOT, XOR, shift
  - Once all major operations are identified for a processor design, **assign binary codes** to each of the operation.
    - For example, say that we want to design a machine that can perform 40 different types of major operations.
    - Then we would require at least 6 bits to represent all of the opcodes.

INST. SET ARCH.

SAMPLE MIPS OPCODES

- Some example opcodes used in the MIPS processors are as follows.

Instruction	Assigned Opcode Value
add $10, $8, $9	**100000**
sub $10, $8, $9	**100010**
and $10, $8, $9	**100100**
or $10, $8, $9	**100101**
lw $10, 0($8)	**100011**
sw $10, 0($8)	**101011**
addi $10, $8, 4	**001000**
nop (no operation)	**000000**

- Note: Different opcodes for **add** and **addi**. Why?

INST. SET ARCH.

CONTROLLER SIGNALS

- Once you have assigned opcodes to all of your major functions, now need to decode the opcodes to the appropriate controller signals.
  - i.e. we no longer want to control the DPU manually.
- Recall that we used the following DPU signals when performing

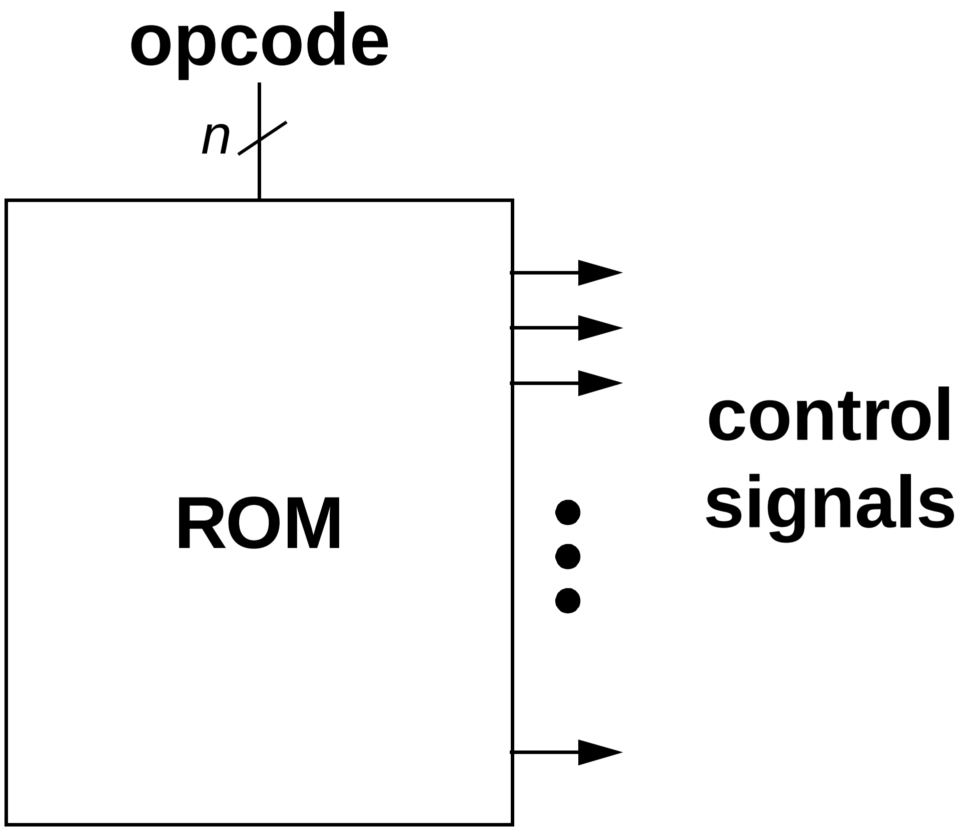
$$\mathbf{R10} = \mathbf{R8} + \mathbf{R9}$$

    - $\bar{\mathbf{a}}/\mathbf{s}$ **= 0** and **en = 1** for **AU**.
    - **en = 0** for **LU**, **en = 0** for **SU**.
    - **st_en = 0**, **ld_en = 0**, **r/w = X**, **msel = 0**.
    - $\mathbf{X_{ra}} = \mathbf{01000}$, $\mathbf{Y_{ra}} = \mathbf{01001}$, $\mathbf{Z_{wa}} = \mathbf{01010}$, and **rwe = 1** for **RF**.
  - Note: We will pass $\mathbf{X_{ra}}$, $\mathbf{Y_{ra}}$, and $\mathbf{Z_{wa}}$ from the outside.
  - Refer to Table 3 in Instruction free-doc for other examples.

INST. SET ARCH.

CONTROLLER SIGNALS

- In general, these control signals can be burned into a ROM.

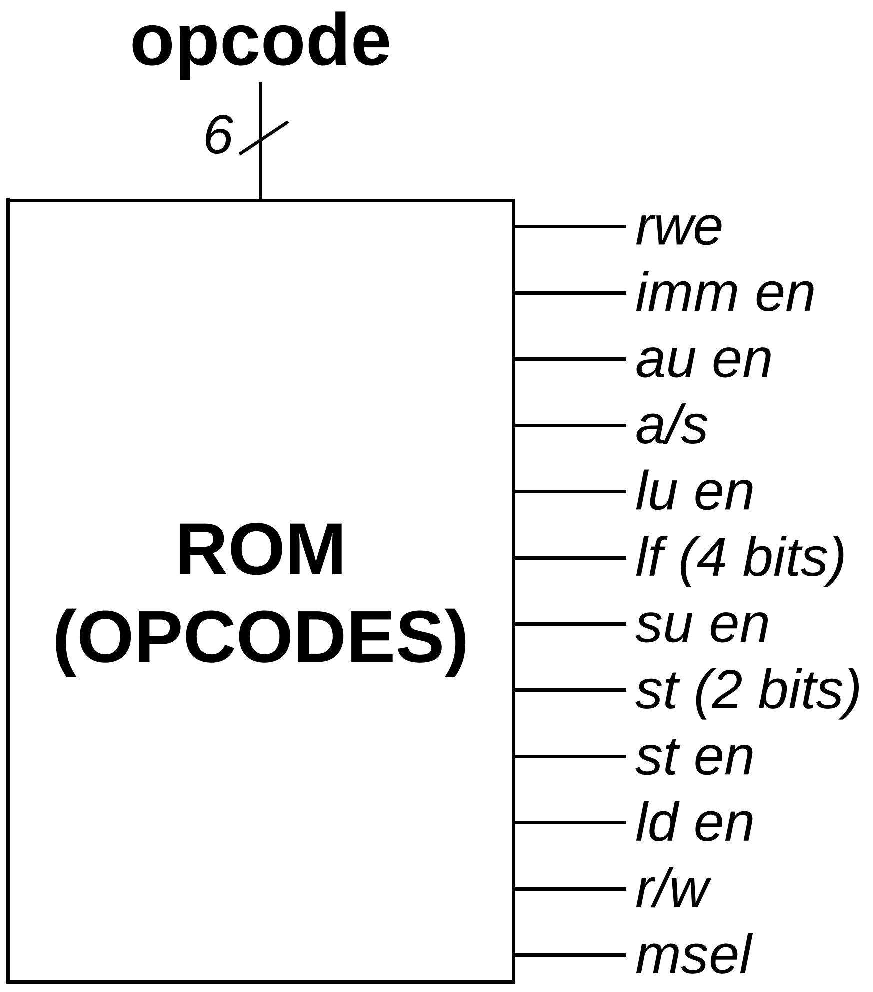


- Each opcode has its own set of general control signals for the DPU.

INST. SET ARCH.

CONTROLLER SIGNALS

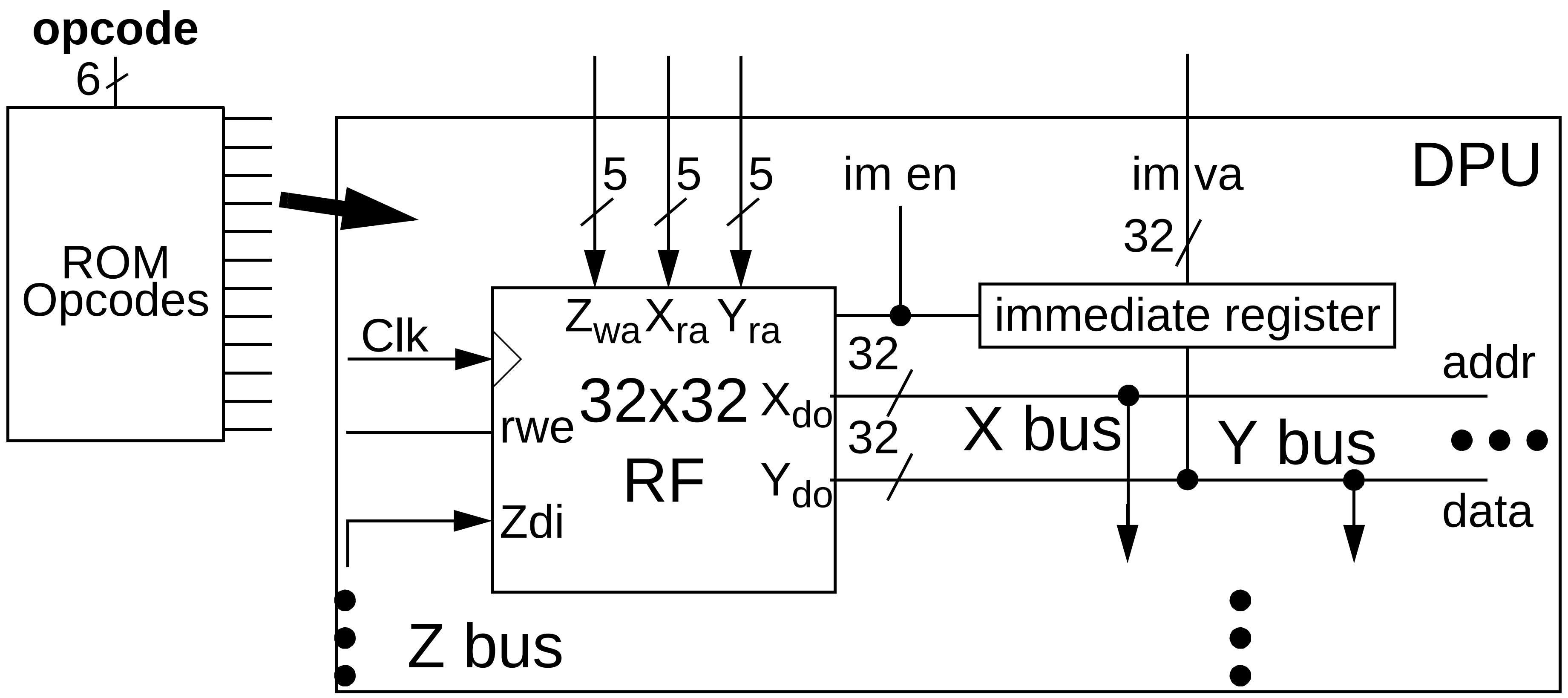
- For our DPU, the control signals are as follows.



INST. SET ARCH.

DPU W/ CONTROLLER

- Now, an input opcode will send appropriate control signals to the DPU for that major operation.

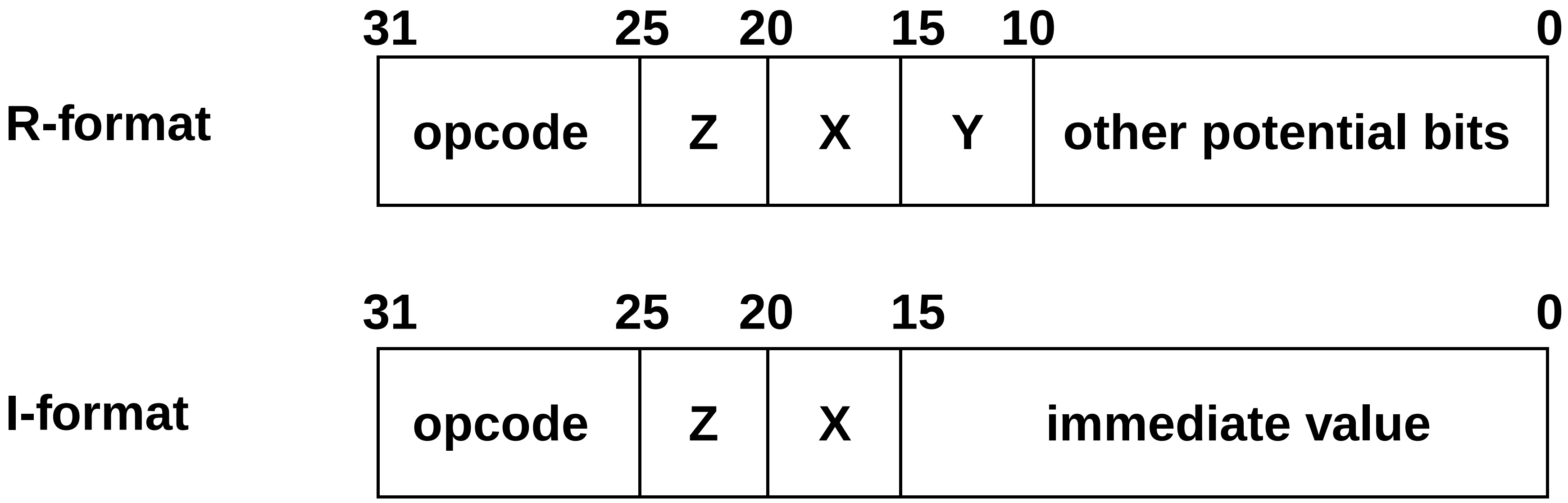


- Notice that we still need register addresses and the immediate value.

INSTRUCTIONS

INSTRUCTION FORMATS

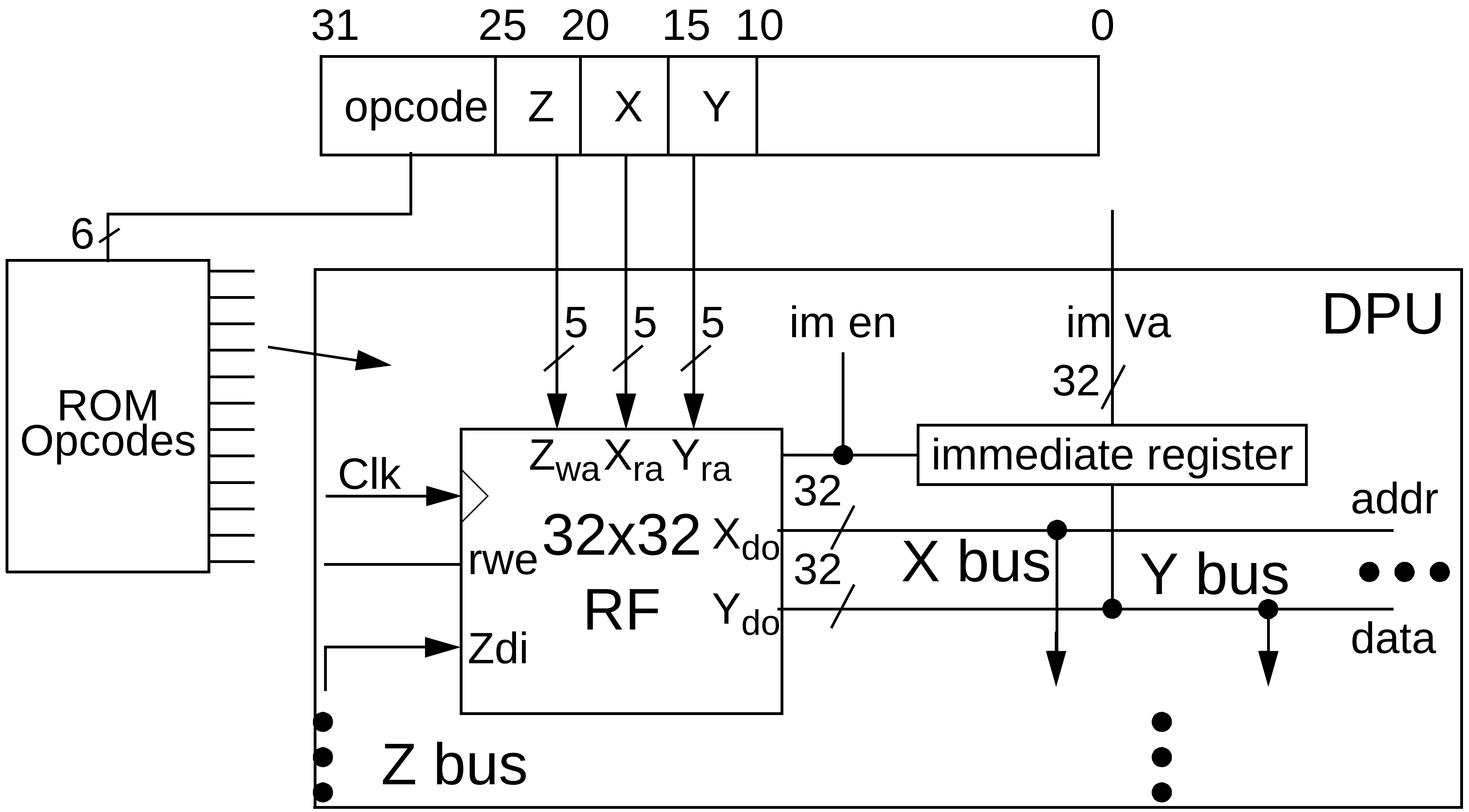
- While instructions can come in many different shapes and forms, we will consider the following 32-bit instruction formats to loosely follow the MIPS R3000/4000 format.



INSTRUCTIONS

R-FORMAT W/ DPU

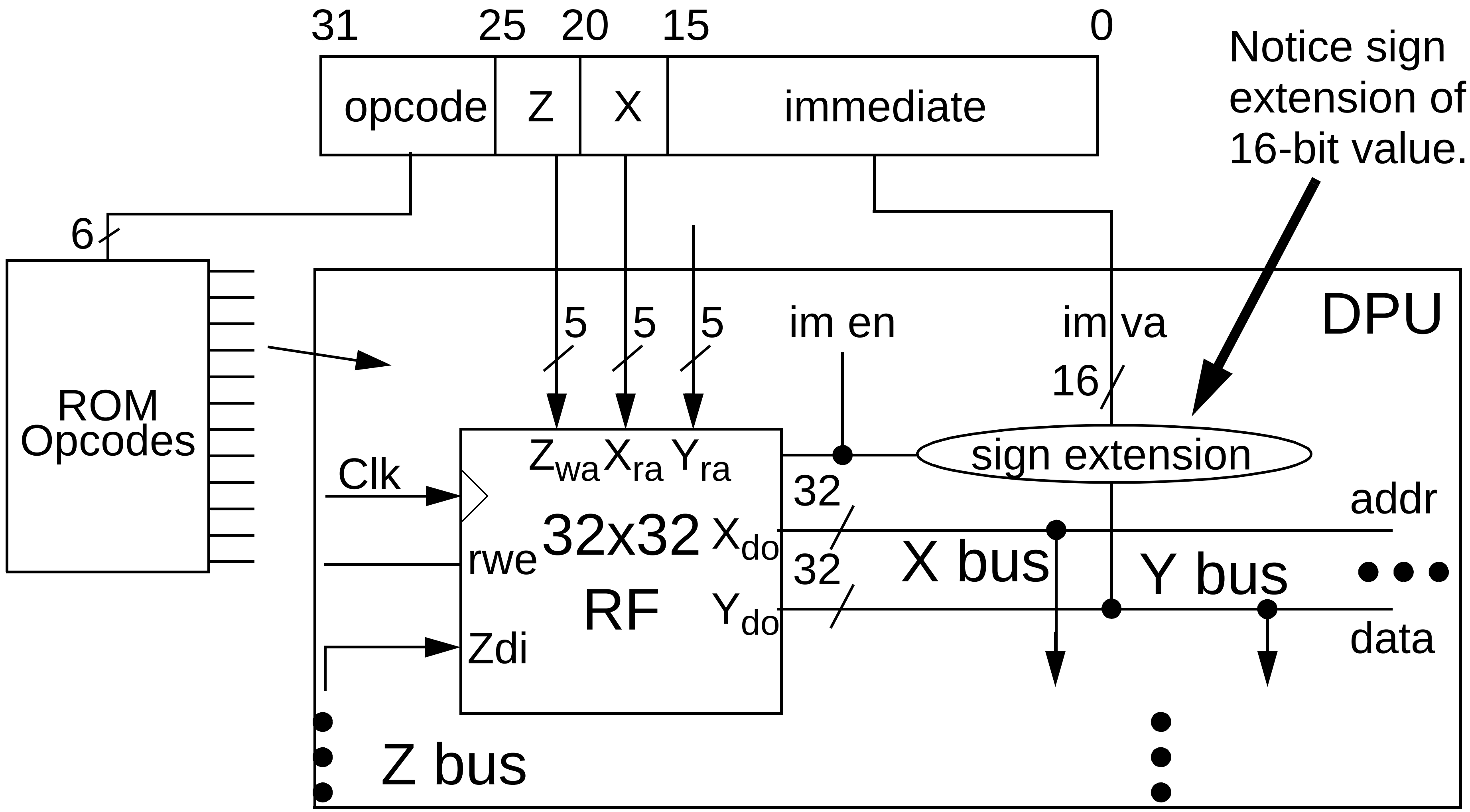
- If we have an R-format instruction, we link the bits as follows.



INSTRUCTIONS

I-FORMAT W/ DPU

- If we have an I-format instruction, we link the bits as follows.

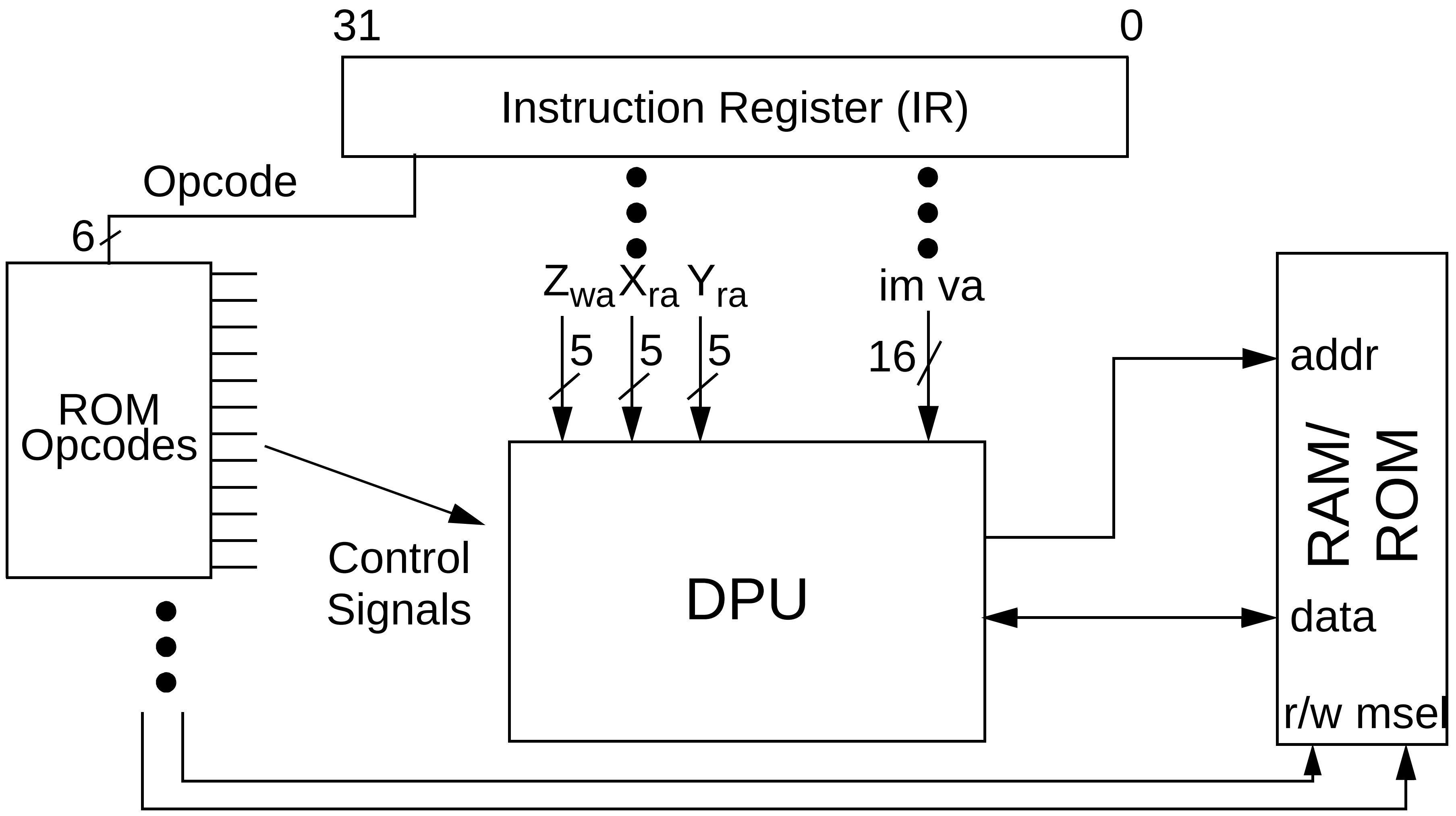


Notice sign extension of 16-bit value.

INSTRUCTIONS

INSTRUCTION REGISTER

- Use a general instruction register that can act as R- or I-Format.


31
0
Instruction Register (IR)
Opcode
6
ROM Opcodes
Z_{wa} X_{ra} Y_{ra}
5
5
5
im va
16
Control Signals
DPU
addr
RAM/ ROM
data
r/w msel