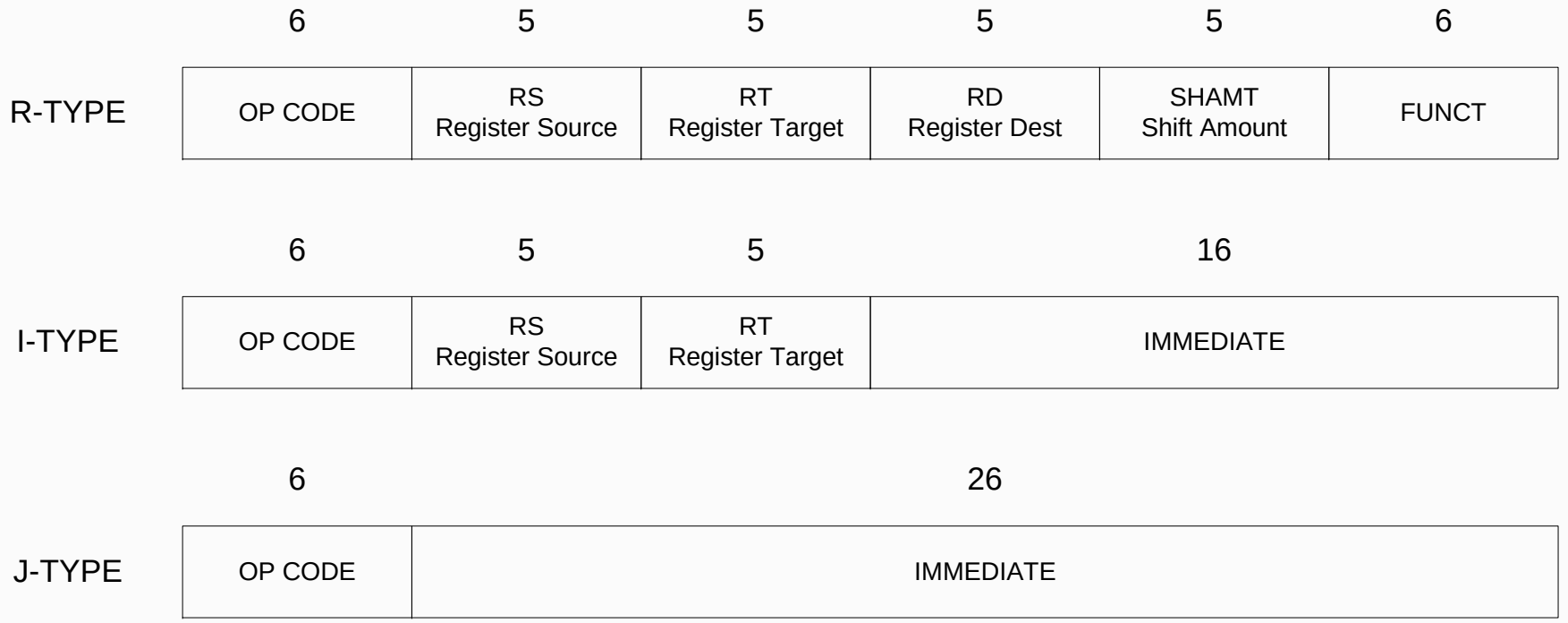
MIPS Instructions

MIPS Details

Overview

- Instruction format



Instruction Format

- J-format: used for j and jal
- I-format: used for instructions with immediates, lw and sw (since the offset counts as an immediate), and the branches (beq and bne),
- (but not the shift instructions; later)
- R-format: used for all other instructions
- It will soon become clear why the instructions have been partitioned in this way.

R-Format Instruction

- opcode: partially specifies what instruction it is (Note: This number is equal to 0 for all R-Format instructions.)
- funct: combined with opcode, this number exactly specifies the instruction
- rs (Source Register): generally used to specify register containing first operand
- rt (Target Register): generally used to specify register containing second operand (note that name is misleading)
- rd (Destination Register): generally used to specify register which will receive result of computation

R-Format Instruction

	6	5	5	5	5	6
R-TYPE	OP CODE	RS Register Source	RT Register Target	RD Register Dest	SHAMT Shift Amount	FUNCT

- shamt: This field contains the amount a shift instruction will shift by. Shifting a 32-bit word by more than 31 is useless, so this field is only 5 bits (so it can represent the numbers 0-31).
- This field is set to 0 in all but the shift instructions.

R-Format examples

- What is the machine code of add $s1, $s3, $s4
- Opcode : 000000 (0 or 0x00)
- RS : 10011 (19 or 0x13)
- RT : 10100 (20 or 0x14)
- RD : 10001 (17 or 0x11)
- SHAMT : 00000
- FUNCT : 100000 (32 or 0x20)
- CODE : 000000 10011 10100 10001 00000 100000
- CODE : 0000 0010 0111 0100 1000 1000 0010 0000
- CODE : 0x02748820

- What is the machine code for sub $t0, $s5, $a0?

R-format example

- What is the machine code for sub $t0, $s5, $a0?
- Opcode : 000000 (0 or 0x00)
- RS : 10101 (21 or 0x15)
- RT : 00100 (4 or 0x04)
- RD : 01000 (8 or 0x08)
- SHAMT : 00000
- FUNCT : 100010 (34 or 0x22)
- CODE : 000000 10101 00100 01000 00000 100010
- CODE : 0000 0010 1010 0100 0100 0000 0010 0010
- CODE : 0x02A44022

- What is the assembly code for 0x02324020

R-format example

- 0x02324020
- 0000 0010 0011 0010 0100 0000 0010 0000
- 000000 10001 10010 01000 00000 100000
- Opcode : 0
- RS : 10001 (17 or 0x11) -> s1
- RT : 10010 (18 or 0x12) -> s2
- RD : 01000 (8 or 0x08) -> t0
- SHAMT : 00000
- Funct : 100000 : 32 or 0x20
- Assembly code: add $t0, $s1, $s2

I-Format Instruction

- opcode: same as before except that, since there's no funct field, opcode uniquely specifies an I-format instruction
- This also answers question of why R-format has two 6-bit fields to identify instruction instead of a single 12-bit field in order to be consistent with other formats.
- rs: specifies the only register operand (if there is one)
- rt: specifies register which will receive result of computation (this is why it's called the target register "rt")

I-Format Instruction

- The Immediate Field:
  - addi, slti, slitu, the immediate is sign-extended to 32 bits. Thus, it's treated as a signed integer.
  - 16 bits can be used to represent immediate up to 2^{16} different values (actually it is $\pm 2^{15}$)
  - This is large enough to handle the offset in a typical lw or sw, plus a vast majority of values that will be used in the slti instruction.

I-Format Problem

- Problem 1:
  - Chances are that addi, lw, sw and slti will use immediates small enough to fit in the immediate field.
  - What if too big?
    - We need a way to deal with a 32-bit immediate in any I-format instruction.
    - New instruction: lui register, immediate
    - stands for Load Upper Immediate
    - takes 16-bit immediate and puts these bits in the upper half (high order half) of the specified register
    - sets lower half to 0s

I-Format Problem

- Example:
  - addi $t0,$t0, 0xABABCDCD
- becomes:
  - lui $at, 0xABAB
  - ori $at, $at, 0xCDCD
  - add $t0,$t0,$at
- Now each I-format instruction has only a 16-bit immediate.
- An instruction that must be broken up is called a pseudoinstruction. (Note that $at was used in this code.)

I-format example

- Machine code for lw $a2, 32($t0)
- Opcode: 0x23 -> 100011
- RS : 0x08 -> 01000
- RT : 0x06 -> 00110
- Immediate value : 32 -> 0x0020 -> 0000 0000 0010 0000
- CODE : 100011 01000 00110 0000 0000 0010 0000
- CODE : 1000 1101 0000 0110 0000 0000 0010 0000
- CODE : 0x8C060020

- What is the code for sw $t4, 1200($s1)?

I-format example

- Machine code for sw $t4, 1200($s1)
- Opcode: 0x2B -> 101011
- RS : 0x11 -> 10001
- RT : 0x0C -> 01100
- Immediate value : 1200 -> 0x04B0 -> 0000 0100 1011 0000
- CODE : 101011 10001 01100 0000 0100 1011 0000
- CODE : 1010 1110 0010 1100 0000 0100 1011 0000
- CODE : 0xAE2C04B0

- What is the code for addi $a0, $a1, 34 (signed)?

I-format example

- Machine code for addi $a0, $a1, 34 (signed)
- Opcode: 0x08 -> 001000
- RS : 0x05 -> 00101
- RT : 0x04 -> 00100
- Immediate value : 34 -> 0x0022 -> 0000 0000 0010 0010
- CODE : 001000 00101 00100 0000 0000 0010 0010
- CODE : 0010 0000 1010 0100 0000 0000 0010 0010
- CODE : 0x20A40022

Branches: PC-Relative Addressing

- opcode specifies beq v. bne
- Rs and Rt specify registers to compare
- What can immediate specify?
  - Immediate is only 16 bits
  - PC is 32-bit pointer to memory
  - So immediate cannot specify entire address to branch to.
- Though we may want to branch to anywhere in memory, a single branch will generally change the PC by a very small amount.

Branches: PC-Relative Addressing

- Solution: PC-Relative Addressing
- Let the 16-bit immediate field be a signed two's complement integer to be added to the PC if we take the branch.
- Now we can branch ± 2^{15} bytes from the PC, which should be enough to cover any loop.
- Note: Instructions are words, so they're word aligned (byte address is always a multiple of 4, which means it ends with 00 in binary).
  - So the number of bytes to add to the PC will always be a multiple of 4.
  - So specify the immediate in words.
- Now, we can branch ± 2^{15} words from the PC (or ± 2^{17} bytes), so we can handle loops 4 times as large.

Branches: PC-Relative Addressing

- Final Calculation:
- If we don't take the branch:
  - PC = PC + 4
- If we do take the branch:
  - PC = (PC + 4) + (immediate * 4)
- Observations
  - Immediate field specifies the number of words to jump, which is simply the number of instructions to jump.
  - Immediate field can be positive or negative.
  - Due to hardware, add immediate to (PC+4), not to PC; will be clearer why later in course

Branch example

- If you have the following code:

```
            bne $s1, $0, label1      0x04000000
            ....                     0x04000004
            ....                     0x04000008
            ....                     0x0400000C
label1:lw ....                       0x04000010
```

- What is the machine code?
- Opcode : 0x05 -> 000101
- RS : 0x11 -> 10001
- RT : 0x00 -> 00000
- label1 is 16 memory location away from bne instruction
  - 16 - 4 = 12
  - 12/4 = 3 (3 instructions away from bne)
  - So the immediate value is 0x0003
- CODE : 0001 0110 0010 0000 0000 0000 0000 0011
- CODE : 0x16200003

Branch example

- If you have the following code:

```
            beq $a1, $s0, label1      0x04000000
            ....
            ....
            ....
label1:lw ....                        0x04000064
```

- What is the machine code?

Branch example

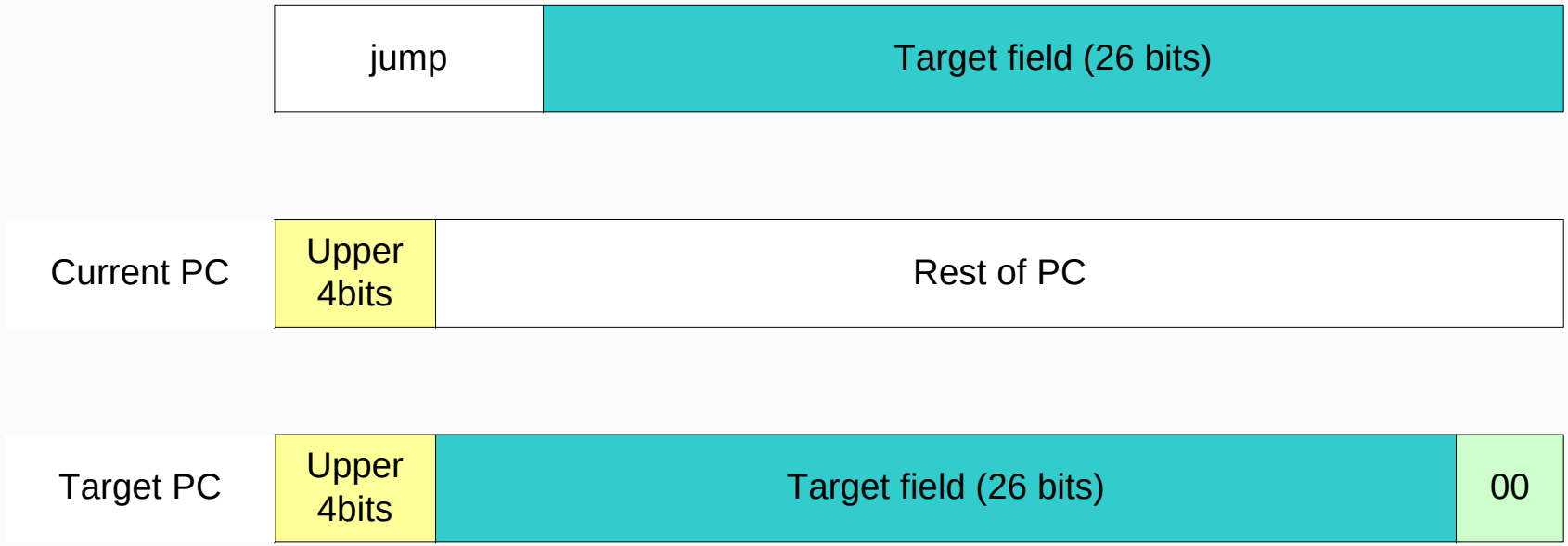
- If you have the following code:

```
            beq $a1, $s0, label1        0x04000000
            ....
            ....
            ....
label1:lw   ....                        0x04000064
```

- What is the machine code?
- Opcode : 0x04 -> 001000
- RS : 0x05 -> 00101
- RT : 0x11 -> 10001
- label1 is 100 memory location away from bne instruction
  - 100 - 4 = 96
  - 96/4 = 24 (24 instructions away from bne)
  - So the immediate value is 0x0018
- CODE : 0010 0000 1011 0001 0000 0000 0001 1000
- CODE : 0x20B10018

Jump Instruction

- Instructions always start on an address that is a multiple of four (they are word-aligned). So the low order two bits of a 32-bit instruction address are always "00". Shifting the 26-bit target left two places results in a 28-bit word-aligned address (the low-order two bits become "00".)
- Now all we need is to fill in the high-order four bits of the address. These four bits come from the high-order four bits in the PC. These are concatenated to the high-order end of the 28-bit address to form a 32-bit address.



Jump example

- If you have the following code:

```
          j label1          0x04000000
          ....              0x04000004
          ....              0x04000008
          ....              0x0400000C
label1:   lw ....           0x04000010
```

- What is the machine code?
- Opcode : 0x02 -> 000010
- label1 address is 0x04000010
  - Remove the upper 4 bits : 0x4000010
  - Remove the last two bits (div by 4) : 0x1000004
- CODE : 0000 1001 0000 0000 0000 0000 0000 0100
- CODE : 0x09000004

Jump example

- If you have the following code:

```
            j label1                    0x04000000
            ....
            ....
            ....
label1:lw ....                          0x04002710
```

- What is the machine code?

Jump example

- If you have the following code:

```
            j label1                    0x04000000
            ....
            ....
            ....
label1:lw ....                          0x04002710
```

- What is the machine code?
- Opcode : 0x02 -> 000010
- label1 address is 0x04002710
  - Remove the upper 4 bits : 0x4002710
  - Remove the last two bits (div by 4) : 0x10009C4
- CODE : 0000 1001 0000 0000 0000 1001 1100 0100
- CODE : 0x0890009C4