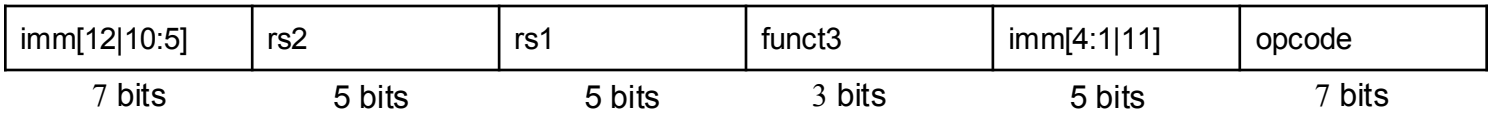
SB/B Format Instructions

imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

beq	SB	Branch EQual	if(R[rs1]==R[rs2) PC=PC+{imm,1b'0}
bge	SB	Branch Greater than or Equal	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}
bgeu	SB	Branch ≥ Unsigned	if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0}
blt	SB	Branch Less Than	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}
bltu	SB	Branch Less Than Unsigned	if(R[rs1]<R[rs2) PC=PC+{imm,1b'0}
bne	SB	Branch Not Equal	if(R[rs1]!=R[rs2) PC=PC+{imm,1b'0}

SB/B Format



- **e.g., beq rs1, rs2, Label**
- **funct3: operation of the instruction**
  - beq= ____, bne=____
- **rs1 and rs2: registers to compare**
- **immediate: see Reference Card**
  - RISC-V supports compressed half-word (16 bits) instructions
  - To accommodate this, all branch offsets (the immediate) are half-word aligned
  - Range is $\pm 2^{11}$ half words, or $\pm 2^{10}$ words
  - What happened to bit-0 of the immediate?

Branch Immediates

- **12 bits, shifted left one bit (multiply by ____)**
  - why?
- **sign extended**
- **added to PC (the branch instruction's address)**
- **Can branch _____ words from PC (_____ bytes)**
- **Why?**
  - branches are used by: `if-else`, `while`, `for`
  - branch often changes PC by a small amount
  - loops are generally small (typically ~50 instructions)

Branch Calculation

- **If we *don't* take the branch:**
  - `PC = PC + 4` (byte address of next instruction)
- **If we *do* take the branch:**
  - `PC = PC + (immediate * 2)`
- **Note:**
  - `immediate` field specifies the number of halfwords to jump
  - `immediate` field is a two's complement value (signed)
  - add `immediate` shifted left 1 bit to PC
  - range:
    - PC + 2*(-2^_____) … PC + 2*(2^_____-1)

Branch Example

opcode=_____

rs1=_____

rs2=_____

immediate=_____

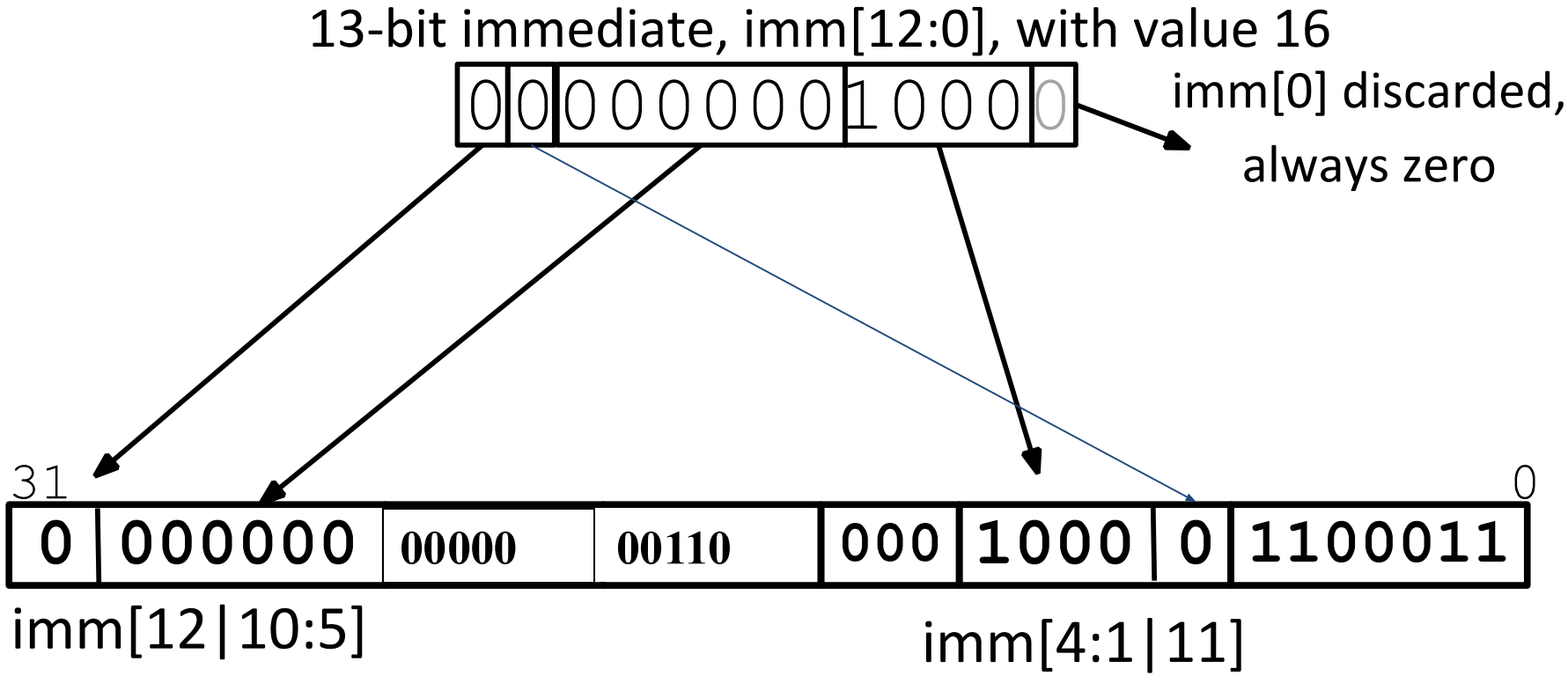
```
Loop:
    beq  t1, zero, End
    add  t0, t0, t2
    addi t1, t1, -1
    jal  zero, Loop
End:
```

- **Branch offset = 4 32-bit instructions = 16 bytes**
- **immediate is number of instructions*2 to add to (<0 to subtract) address of branch instruction.**

imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
???????	00000	00110	000	?????	1100011

Branch Example

beq t1, zero, offset = 16 bytes



hex 0x __ __ __ __ __ __ __ __

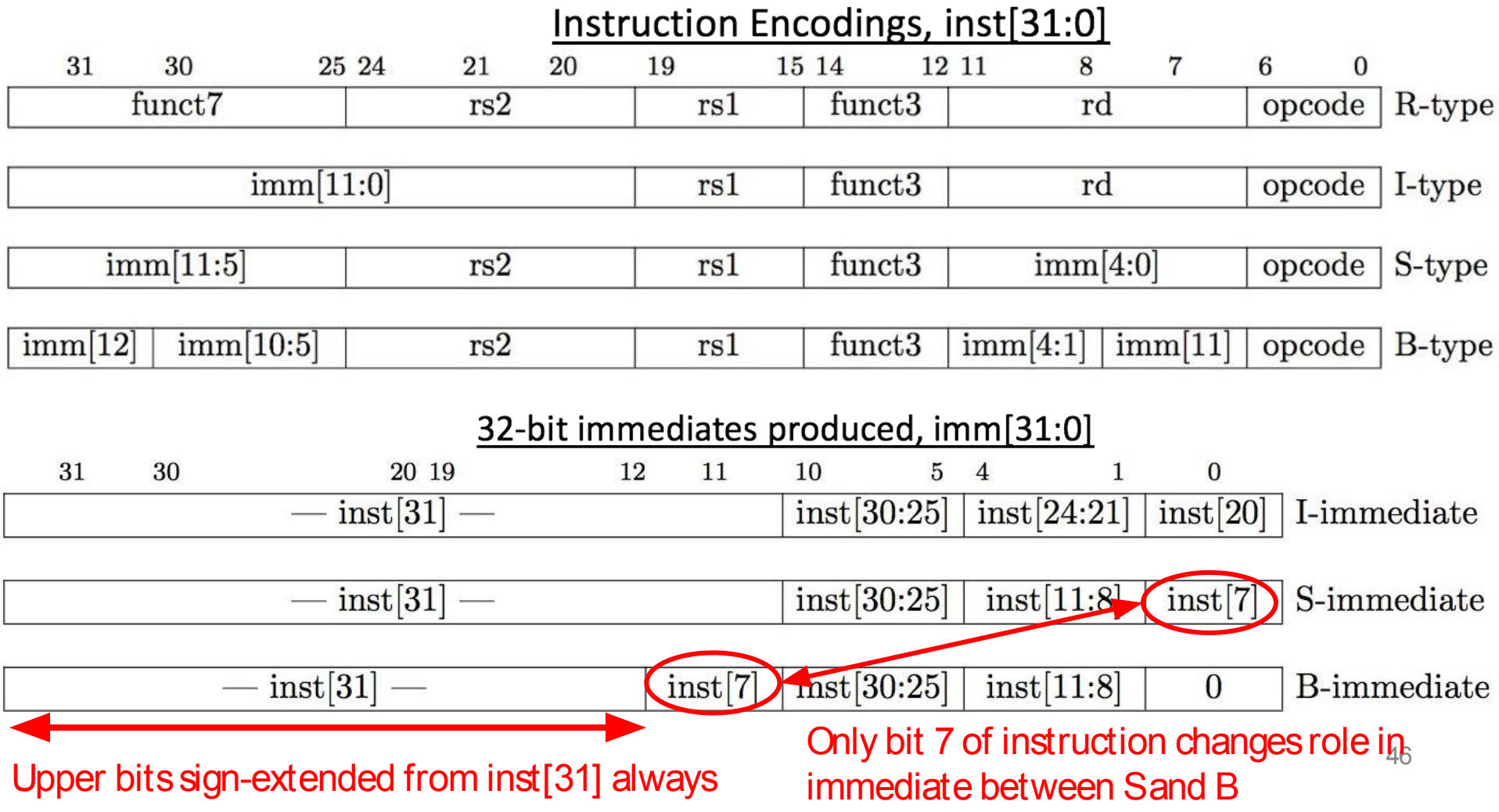
Why are the Addresses so Tangled up for SB?

imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode

- **Because shifting costs space on silicon.**
- **With this layout, only bit 11 has to be shifted, and a 0 put in its place, relative to the S-format immediate value handling**

- **Why not put bit 12 down in the 0 position?**
  - Make it easy for sign extension

Why is it so Confusing?!



Unconditional Branches/Jumps

- **Conditional branches**
  - assume we won't branch very far
  - can specify changes in PC
- **Unconditional branch (jal)**
  - may jump *anywhere* in memory (a 32-bit address)
  - can't fit both a 7-bit opcode and a 32-bit address in the 32-bit instruction
  - compromise (good design)

UJ-Format (i.e., J Format)

immediate[20|10:1|11|19:12]	rd	opcode

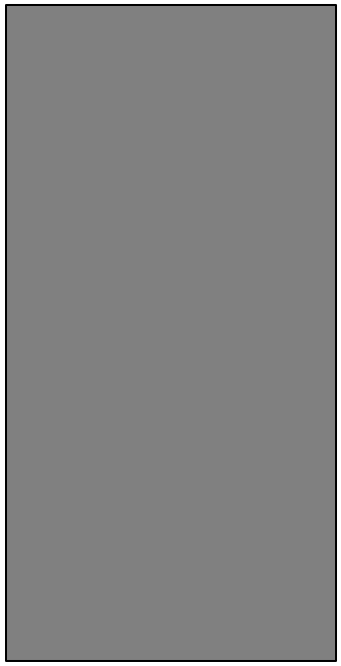
jal	UJ	Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}

- **opcode format identical to R-format and I-format**
  - consistency simplifies hardware
- **Rest of instruction used for target address**
  - RISC-V supports compressed half-word (16 bits) instructions
  - To accommodate this, all branch offsets (the immediate) are half-word aligned (all addresses are even … bit 0 is always 0)
  - What happened to bit-0 of the immediate?
- **How is address calculated? Untangled and shift left 1 bit**

How Far Can jal Jump?

- **Range is _____ instructions**
  - 2^20 values since 20 bits
  - *2 since lowest bit is an assumed 0 to get # bytes
  - /4 since 4 bytes per instruction
- **Where in that range do we start? (trick question)**

- **Jump only inadequate if needs to straddle a 2^{18} boundary. It works 99.999…% of the time, since programs usually aren't that long.**
- **Technically cannot jump *anywhere* in memory.**



UJ-Format Example

```
0x00400000 Loop: slli t1, s3, 2
0x00400004       …
0x00400008       …
0x0040000C       …
0x00400010       …
0x00400014 jal zero, Loop
0x00400018       …
```

opcode = _____

PC = _____

PC@Loop = _____

PC=PC+{imm,1b'0}=0x00400014+0xFFFEC (2's complement)

0xFFFEC=0x1 1111 1111 1111 1110 1100 (21 bits)

immediate[20|10:1|11|19:12]	rd	opcode
11111110110111111111	00000	1101111

Don't stereotype!

- **Why isn't jalr a UJ-type instruction?**

- **What is the difference between `jal` and `jalr`?**

Branching Far Away

- **What if conditional branch must jump further?**

```
beq t0, t1, L1    # L1 >> 2^10-1
```

can become:

```
beq t0, t1, L2
    ...
  jal zero, L3 # jump around the re-jump
L2: jal zero, L1 # within 2^18 from here
L3: ...          # continue with code
```

All Instruction Formats

- **6 basic types of instruction formats in RISC-V**
  - R-format: (“register”)
    - math and logic: add, sub, and, or, sll, sra
  - I-format: (“immediate”)
    - lw, lb, slli, srli, addi, andi, ori, xori, jalr
  - S-format (“store”)
    - sw, sb
  - SB-format (“branch”) or B-format
    - beq, bne, bge, blt, bgeu, bltu
  - UJ-format (“jump”) or J-format
    - jal
  - U-format (instructions with “upper immediates”)
    - lui, auipc

RISC-V Instruction Format Summary

- **Minimize number of instructions**
  - 6 types
  - Instructions have 1, 2, or 3 operands

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

After-Class Activities

- **Take a few instructions from your last program and generate the machine language for them by hand.**
- **Compare your results to the assembled results in RARS.**