_____ _____

Your Name (first last) *SID*

UC Berkeley CS61C
Final Exam Fall 2019

_____ _____ _____

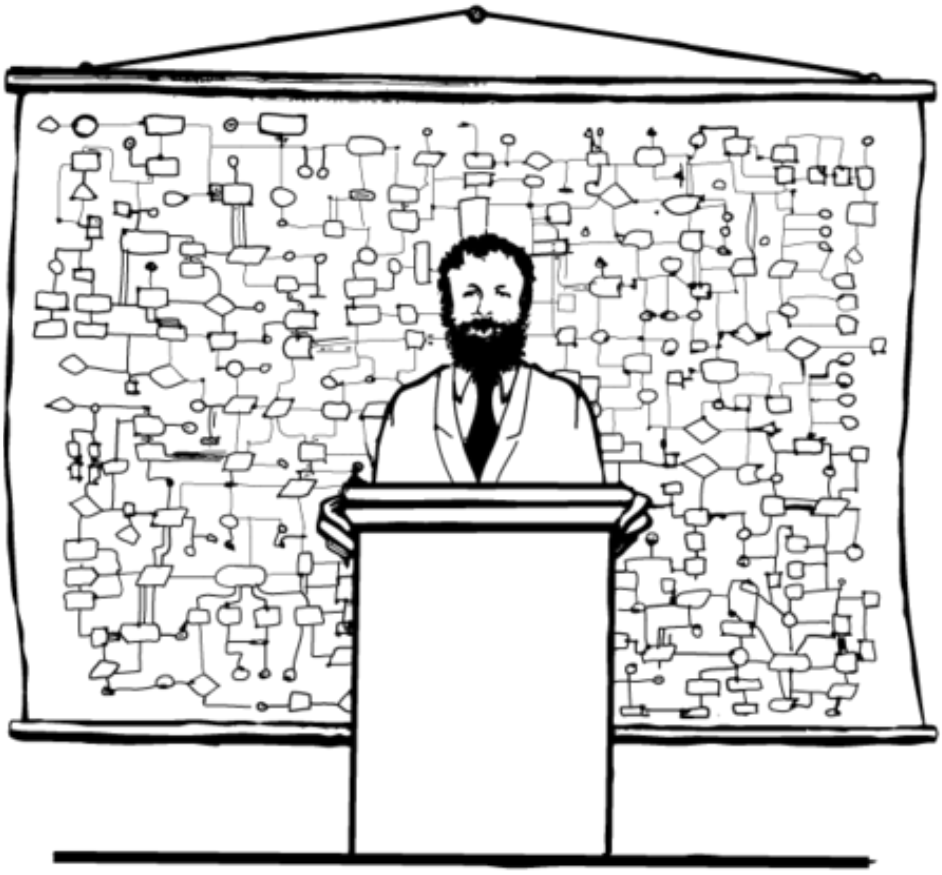
← Name of person on left (or aisle) *TA name* *Name of person on right (or aisle) →*

Fill in the correct circles & squares completely…like this: ● (select ONE), and ■(select ALL that apply)

When you see SHOW YOUR WORK, that means a correct answer *without work* will receive NO CREDIT, and your work needs to show how you were led to the answer you reached. If you find that there are multiple correct answers to a "select ONE" question, please choose just one of them.

Question	1	2	3	4	5	6	7	8	9	10	Total
Minutes	2	8	20	30	30	12	18	15	15	30	**180**
Points	5	11	14	30	30	12	18	15	15	30	**180**

Quest-clobber questions: Q2ad, Q3
Midterm-clobber questions: Q1-6



"Now that you have an overview of the system, we're ready for a little more detail"

CS61C Final Clarifications

1. All answers should be fully simplified unless otherwise stated.
2. Value of the float, not the bits
3. ASCII Value 0xFF == nbsp
4. RISCV Qa: 0 < x < 10.

Q1) CALLer/CALLee Convention… (5 pts)

Determine which stage of **CALL** each of the following actions happen in. Select ONE per row.	**C**ompiler	**A**ssembler	**L**inker	**L**oader
a) Copying a program from the disk into physical memory	◯	◯	◯	◯
b) Removing pseudoinstructions	◯	◯	◯	◯
c) Determining increment size for pointer arithmetic	◯	◯	◯	◯
d) Incorporating statically-linked libraries	◯	◯	◯	◯
e) Incorporating dynamically-linked libraries	◯	◯	◯	◯

Q2) Open to Interpretation (11 pts = 2 + 3 + 4 + 2)

Let's consider the hexadecimal value **`0xFF000003`**. How is this data interpreted, if we treat this number as…

a) an array `A` of unsigned, 8-bit numbers? Please write each number in **decimal,** assume the machine is **big endian**, and write **`A[0]`** on the left, **`A[3]`** on the right.

_____, _____, _____, _____

SHOW YOUR WORK HERE

b) an IEEE-754 single-precision floating point number?

SHOW YOUR WORK

c) a RISC-V instruction? If there's an immediate, write it in decimal.

_____

SHOW YOUR WORK

d) a **`(uint32_t *)`** variable **`c`** in **little-endian** format, and we call **`printf((char *) &c)`**? If an error or undefined behavior occurs, write "Error". If nothing is printed, write "Blank". Please refer to the ASCII table provided on your reference sheet. For non-printable characters, please write the value in the Char column from the table. For example, for a single backspace character, you would write "**BS".**

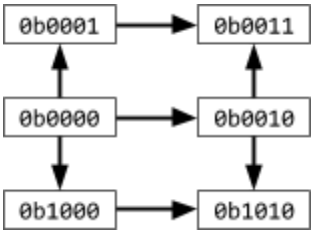
_____

SHOW YOUR WORK

Q3) There's a Dr. Hamming to C you... (14 pts)

We are given an array of **N unique** `uint32_t` that represent nodes in a directed graph. We say there is an edge between **A** and **B** if **A < B** and the Hamming distance between A and B is **exactly 1**. A Hamming distance of 1 means that the bits differ in 1 (and only 1) place. As an example, if the array were {`0b0000, 0b0001, 0b0010, 0b0011, 0b1000, 0b1010`}, we would have the edges shown on the right:

A	B
0b0000	0b0001
0b0000	0b0010
0b0000	0b0100
0b0001	0b0011
0b0010	0b0011
0b0010	0b1010
0b1000	0b1010



Construct an `edgelist_t` (specified below) that contains all of the edges in this graph. Our solution used every line provided, but if you need more lines, just write them to the right of the line they're supposed to go after and put semicolons between them. All of the necessary `#include` statements are omitted for brevity; don't worry about checking for `malloc`, `calloc`, or `realloc` returning NULL. *Make sure `L->edges` has no unused space when L is eventually returned.*

```
typedef struct {
   uint32_t A;
   uint32_t B;
} edge_t;

typedef struct {
   edge_t *edges;
   int len;
} edgelist_t;
```

```
edgelist_t *build_edgelist(uint32_t *nodes, int N) {
   edgelist_t *L = (edgelist_t *) malloc (sizeof(edgelist_t));
   L->len   = 0;


   L->edges = ____________________________________________________________________


   for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {


           uint32_t tmp = ____________________________________________________________


           if (_____________________________________________________________________) {


               ____________________________________________________________________


               ____________________________________________________________________
               L->len++;
           }
       }
   }


   ____________________________________________________________________________
   return L;
}
```

Q4) Felix Unger must have written this RISC-V code! **(30 pts = 3*10)**

```
mystery:
      la t6, loop
loop: addi x0, x0, 0        ### nop
      lw   t5, 0(t6)
      addi t5, t5, 0x80
      sw   t5, 0(t6)
      addi a0, a0, -1
      bnez a0, loop
      ret
```

You are given the code above, and told that you can read and write to any word of memory without error. The function **mystery** lives somewhere in memory, but *not* at address **0x0**. Your system has no caches.

a) At a functional level, in seven words or fewer, what does **mystery(x)** do when **x < 10**?

_____ _____ _____ _____ _____ _____ _____

b) One by one, what are the values of **a0** that **bnez** sees with **mystery(13)** at every iteration? We've done the first few for you. List no more than 13; if it sees fewer than 13, write N/A for the rest.

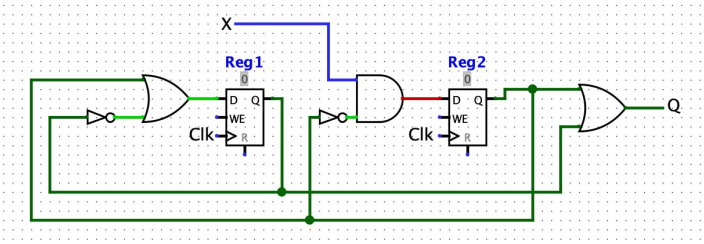
12, 11, _____, _____, _____, _____, _____, _____, _____, _____, _____, _____, _____

c) How many times is the **bnez** instruction seen when **mystery(33)** is called before it reaches **ret** (if it ever does)? If it's infinity, write ∞. _____

d) Briefly (two sentences max) explain your answer for part (c) above.

Q5) Watch the clock and don't delay! **(30 pts = 2*5 + 10 + 10)**

Consider the following circuit:



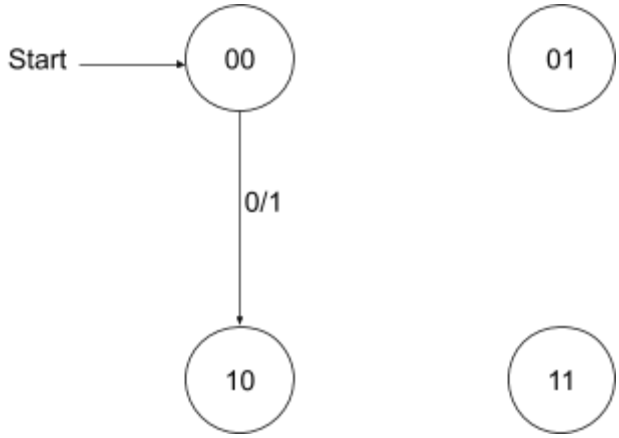
You are given the following information:

- Clk has a frequency of 50 MHz
- AND gates have a propagation delay of 2 ns
- NOT gates have a propagation delay of 4 ns
- OR gates have a propagation delay of 10 ns
- X changes 10ns after the rising edge of Clk
- Reg1 and Reg2 have a clock-to-Q delay of 2 ns

SHOW YOUR WORK BELOW

a) What is the **longest possible setup time** such that there are no setup time violations? **_____ ns**	
b) What is the **longest possible hold time** such that there are no hold time violations? **_____ ns**	

c) Represent the circuit above using an equivalent FSM, where X is the input and Q is the output, with the state labels encoding Reg1Reg2 (e.g., "01" means Reg1=0 and Reg2=1). We did one transition already.



d) Draw the **FULLY SIMPLIFIED** (*fewest* primitive gates) circuit for the equation below into the diagram on the lower right. You may use the following primitive gates: AND, NAND, OR, NOR, XOR, XNOR, and NOT.

$$out = \overline{(C + AB\overline{C} + \overline{B}\,\overline{C}D)} + \overline{(C + \overline{B} + D)}$$

SHOW YOUR WORK IN THIS BOX

Q6) RISCV Exam-isim Debug – Single Cycle **(12 pts = 2 x 6)**

For your CPU project, you followed the datapath diagram we gave you exactly and built a single-cycle CPU. However, something is not working correctly. *All instructions besides some of the I-types and SB-types are working.* You start by testing with an **`addi a0 a0 -3`** instruction. The **`a0`** register initially holds a value of **`7`** and all other registers initially hold **`0`**. This instruction is stored in IMEM at address **`0x00000004`**. DMEM reflects the initial IMEM. Undefined ImmSel outputs an I-type imm. You put a probe at the data read from IMEM and find the instruction is correct. You next put a probe at **`wb`**, and see the output is `0b0000_0000_0000_0000_0001_0000_0000_0100` (**`0x00001004`**).

a) Since the output is incorrect, what errors alone would cause the erroneous behavior? (select all that apply)

- ☐ The RegWEn is set to false.
- ☐ The Immediate Generator is not sign extending.
- ☐ Read Reg1 and Read Reg2 are flipped.
- ☐ The writeback MUX is selecting DMEM.
- ☐ PCSel is set to PC + 4.
- ☐ The writeback MUX is selecting PC + 4.
- ☐ MemRW is set to write.
- ☐ BSel is selecting rs2 and not imm.

You fix that issue. You then test **`beq x0 a0 label`** but something is still not working. This instruction is at address **`0xbfffff00`** and **`label`** is at address **`0xbfffff40`**. The register **`a0`** holds **`0`** and all other registers hold **`1`**. Assume that we get the correct instruction machine code for **`beq x0 a0 label`** when we probe it. You put a probe before the PC register and see this incorrect output: **`0xbfffff20.`**
Note: All other instruction types are working.

b) What errors alone would cause the erroneous behavior? (select all that apply)

- ☐ WBSel is incorrect.
- ☐ ImmSel is Incorrect.
- ☐ PCSel is set to PC + 4.
- ☐ There is an error in the Read Data1/rs1 wire.
- ☐ The ImmGen is not correctly padding w/ extra 0.
- ☐ The inputs to the ASel MUX are flipped.
- ☐ The inputs to the BSel MUX are flipped.
- ☐ Read Reg1 and Read Reg2 are flipped.

Q7) *RISCV Exam-isim Debug – Pipelined* (18 pts = 3 + 6 + 3 + 6)

After solving your datapath bug, you decide to introduce the traditional five-stage pipeline into your processor. You find that your unit tests with single commands work for all instructions, and write some test patterns with multiple instructions. After running the test suite, the following cases fail. You should assume registers are initialized to 0, the error condition is calculated in the fetch stage, and no forwarding is currently implemented.

Case 1: Assume the address of an array with all different values is stored in `s0`.

```
addi  t0 x0 1
slli  t1 t0 2
add   t1 s0 t1
lw    t2 0(t1)
```

Each time you run this test, there is the same incorrect output for `t2`. All the commands work individually on the single-stage pipeline.
Pro tip: you shouldn't even need to understand what the code does to answer this.

a) What caused the failure? (select ONE)	**b)** How could you fix it? (select all that apply)
◯ Control Hazard ◯ Structural Hazard ◯ Data Hazard ◯ None of the above	☐ Insert a nop 3 times if you detect this specific error condition ☐ Forward execute to write back if you detect this specific error condition ☐ Forward execute to memory if you detect this specific error condition ☐ Forward execute to execute if you detect this specific error condition ☐ Flush the pipeline if you detect this specific error condition

Case 2: *After fixing that hazard,* the following case fails:

```
      addi  s0 x0 4
      slli  t1 s0 2
      bge   s0 x0 greater
      xori  t1 t1 -1
      addi  t1 t1 1
greater:
      mul t0 t1 s0
```

When this test case is run, `t0` contains `0xFFFFFFC0`, which is not what it should have been.
Pro tip: you shouldn't even need to understand what the code does to answer this.

c) What caused the failure? (select ONE)	**d)** How could you fix it? (select all that apply)
◯ Control Hazard ◯ Structural Hazard ◯ Data Hazard ◯ None of the above	☐ Insert a nop 3 times if you detect this specific error condition ☐ Forward execute to write back if you detect this specific error condition ☐ Forward execute to memory if you detect this specific error condition ☐ Forward execute to execute if you detect this specific error condition ☐ Flush the pipeline if you detect this specific error condition

Q8) This is for all the money! (15 pts = 3 + 7 + 5)

Assume we have a single-level, 1 KiB direct-mapped L1 cache with 16-byte blocks. We have 4 GiB of memory. An integer is 4 bytes. The array is block-aligned.

```
#define LEN 2048

int ARRAY[LEN];
int main() {
    for (int i = 0; i < LEN - 256; i+=256) {
       ARRAY[i] = ARRAY[i] + ARRAY[i+1] + ARRAY[i+256];
       ARRAY[i] += 10;
    }
}
```

a) Calculate the number of tag, index, and offset **bits** in the L1 cache.

T:___	I:___	O:___

SHOW YOUR WORK

b) What is the hit rate for the code above? Assume C processes expressions left-to-right.

_____

SHOW YOUR WORK

c) You decide to add an L2 cache to your system! You shrink your L1 cache, so it now takes 3 cycles to access L1. Since you have a larger L2 cache, it takes 50 cycles to access L2. The L1 cache has a hit rate of 25% while the L2 cache has a hit rate of 90%. It takes 500 cycles to access physical memory. What is the average memory access time in cycles?

_____

SHOW YOUR WORK

Q9) We've got VM! Where? **(15 pts = 2 + 3 + 5 + 5*1)**

Your system has a 32 TiB virtual address space with a single level page table. Each page is 256 KiB. On average, the probability of a TLB miss is 0.2 and the probability of a page fault is 0.002. The time to access the TLB is 5 cycles and the time to transfer a page to/from disk is 1,000,000 cycles. The physical address space is 4 GiB and it takes 500 cycles to access it. The system has an L1 physically indexed and tagged cache which takes 5 cycles to access and a hit rate of 50%. On a TLB miss, the MMU checks physical memory next.

a) How many bits is the Virtual Page Number?

_____ bits

SHOW YOUR WORK

b) What is the total size of the page table (in **bits**), assuming we have **no** permission bits or any other metadata in a page table entry, just the translation?

_____ bits

SHOW YOUR WORK

c) What is the average memory access time (in cycles) for a single memory access for the current process? Assume the page table is resident in DRAM.

_____ cycles

SHOW YOUR WORK

d) Which of the following, if any, **must be done** when we switch to a different process? Do **not** select any option that is unnecessary.

		Yes	No
1)	Update page table address register	◯	◯
2)	Evict pages for the previous process from RAM	◯	◯
3)	Clear TLB dirty bits	◯	◯
4)	Clear cache valid bits	◯	◯
5)	Clear TLB valid bits	◯	◯

Q10) Parallelism and Potpourri (30 pts = 6 + 4 + 3 + 3 + 3 + 3 + 4 + 4)

a) What are all the possible values of `x` and `y` after execution has completed if the code were run on two cores concurrently?

x: ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4
☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9

y: ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4
☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9

```
int x = 1;
int y = 1;
#pragma omp parallel
{
   x += 1;
   y = x + y;
}
```

SHOW YOUR WORK

b) A Job involves four Tasks, and the % of time spent in each Task is shown in the table. If we buy accelerators that speed up `f` by 2x and `k` by 8x what's your *total speedup*?

_____

Task	%
f	10% → 2x
g	4%
h	6%
k	80% → 8x

SHOW YOUR WORK

c) Which of the following were discussed in the MapReduce lecture? (select all that apply)

☐ Workers specialize: A "map" worker that finishes their map task early are *only* given a new map task.
☐ The system automatically reassigns tasks if a worker "dies", providing automatic fault-tolerance.
☐ MapReduce was specifically designed for custom high-end machines and custom high-end networks.
☐ Hadoop was better than Spark, since Hadoop offered better performance, lazy evaluation, and interactivity.

d) Virtual memory allows us to: (select all that apply)

☐ Pretend that programs do not have to share the address space with other programs.
☐ Have more stable and secure computer systems.
☐ Divide the entire address space into 4 sections specifically for static, code, heap, and stack.
☐ Provide the illusion that the computer has access to storage the size of DRAM but at the speed of disk.

e) Which of the following were discussed in Prof. David Patterson's lecture? (select all that apply)

☐ The power demands of machine learning are growing 10x per year; Moore's Law is only 10x in 5 years.
☐ The Tensor Processing Unit has a similar performance per watt to a CPU on production applications.
☐ The marketplace has decided: Open ISAs (e.g., RISC-V) are better than proprietary ones (e.g., ARM).
☐ Domain Specific Architectures achieve incredible performance but just for one application, like ASICs.

f) Which of the following were discussed in James Percy's GPU lecture? (select all that apply)

☐ A square is the base shape used when rendering scenes.
☐ The GPU achieves its speed because all of the threads run different programs on the same data.
☐ A GPU has many more cores than a CPU and operates at a higher frequency.
☐ When pixels along polygon edges are different between new generations of GPUs, the team investigates it.

g) You have an SSD which can transfer data in 32-byte chunks at a rate of 64 MB/second. No transfer can be missed. If we have a 4GHz processor, which takes 200 cycles for a polling operation, what fraction of time does the processor spend polling the SSD drive for data? Leave your answer in the box provided as a percentage.

SHOW YOUR WORK

h) You are designing a 64-bit ISA for a simplified CPU with 3 bit-fields: `immediate | register | opcode`. You reserve enough of the rightmost bits to handle 1,500 opcodes, and enough of the leftmost bits to encode unsigned numbers up to 500 trillion. What's the greatest number of registers can you have?

SHOW YOUR WORK

RV64I BASE INTEGER INSTRUCTIONS, in alphabetical order

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	NOTE
add,addw	R	ADD (Word)	R[rd] = R[rs1] + R[rs2]	1)
addi,addiw	I	ADD Immediate (Word)	R[rd] = R[rs1] + imm	1)
and	R	AND	R[rd] = R[rs1] & R[rs2]	
andi	I	AND Immediate	R[rd] = R[rs1] & imm	
auipc	U	Add Upper Immediate to PC	R[rd] = PC + {imm, 12'b0}	
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}	2)
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}	2)
bne	SB	Branch Not Equal	if(R[rs1]!=R[rs2) PC=PC+{imm,1b'0}	
ebreak	I	Environment BREAK	Transfer control to debugger	
ecall	I	Environment CALL	Transfer control to operating system	
jal	UJ	Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}	
jalr	I	Jump & Link Register	R[rd] = PC+4; PC = R[rs1]+imm	3)
lb	I	Load Byte	R[rd] = {56'bM[](7),M[R[rs1]+imm](7:0)}	4)
lbu	I	Load Byte Unsigned	R[rd] = {56'b0,M[R[rs1]+imm](7:0)}	
ld	I	Load Doubleword	R[rd] = M[R[rs1]+imm](63:0)	
lh	I	Load Halfword	R[rd] = {48'bM[](15),M[R[rs1]+imm](15:0)}	4)
lhu	I	Load Halfword Unsigned	R[rd] = {48'b0,M[R[rs1]+imm](15:0)}	
lui	U	Load Upper Immediate	R[rd] = {32b'imm<31>, imm, 12'b0}	
lw	I	Load Word	R[rd] = {32'bM[](31),M[R[rs1]+imm](31:0)}	4)
lwu	I	Load Word Unsigned	R[rd] = {32'b0,M[R[rs1]+imm](31:0)}	
or	R	OR	R[rd] = R[rs1] | R[rs2]	
ori	I	OR Immediate	R[rd] = R[rs1] | imm	
sb	S	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)	
sd	S	Store Doubleword	M[R[rs1]+imm](63:0) = R[rs2](63:0)	
sh	S	Store Halfword	M[R[rs1]+imm](15:0) = R[rs2](15:0)	
sll,sllw	R	Shift Left (Word)	R[rd] = R[rs1] << R[rs2]	1)
slli,slliw	I	Shift Left Immediate (Word)	R[rd] = R[rs1] << imm	1)
slt	R	Set Less Than	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	
slti	I	Set Less Than Immediate	R[rd] = (R[rs1] < imm) ? 1 : 0	
sltiu	I	Set < Immediate Unsigned	R[rd] = (R[rs1] < imm) ? 1 : 0	2)
sltu	R	Set Less Than Unsigned	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	2)
sra,sraw	R	Shift Right Arithmetic (Word)	R[rd] = R[rs1] >> R[rs2]	1,5)
srai,sraiw	I	Shift Right Arith Imm (Word)	R[rd] = R[rs1] >> imm	1,5)
srl,srlw	R	Shift Right (Word)	R[rd] = R[rs1] >> R[rs2]	1)
srli,srliw	I	Shift Right Immediate (Word)	R[rd] = R[rs1] >> imm	1)
sub,subw	R	SUBtract (Word)	R[rd] = R[rs1] – R[rs2]	1)
sw	S	Store Word	M[R[rs1]+imm](31:0) = R[rs2](31:0)	
xor	R	XOR	R[rd] = R[rs1] ^ R[rs2]	
xori	I	XOR Immediate	R[rd] = R[rs1] ^ imm	

OPCODES IN NUMERICAL ORDER BY OPCODE

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIMAL
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2
ld	I	0000011	011		03/3
lbu	I	0000011	100		03/4
lhu	I	0000011	101		03/5
lwu	I	0000011	110		03/6
addi	I	0010011	000		13/0
slli	I	0010011	001	0000000	13/1/00
slti	I	0010011	010		13/2
sltiu	I	0010011	011		13/3
xori	I	0010011	100		13/4
srli	I	0010011	101	0000000	13/5/00
srai	I	0010011	101	0100000	13/5/20
ori	I	0010011	110		13/6
andi	I	0010011	111		13/7
auipc	U	0010111			17
addiw	I	0011011	000		1B/0
slliw	I	0011011	001	0000000	1B/1/00
srliw	I	0011011	101	0000000	1B/5/00
sraiw	I	0011011	101	0100000	1B/5/20
sb	S	0100011	000		23/0
sh	S	0100011	001		23/1
sw	S	0100011	010		23/2
sd	S	0100011	011		23/3
add	R	0110011	000	0000000	33/0/00
sub	R	0110011	000	0100000	33/0/20
sll	R	0110011	001	0000000	33/1/00
slt	R	0110011	010	0000000	33/2/00
sltu	R	0110011	011	0000000	33/3/00
xor	R	0110011	100	0000000	33/4/00
srl	R	0110011	101	0000000	33/5/00
sra	R	0110011	101	0100000	33/5/20
or	R	0110011	110	0000000	33/6/00
and	R	0110011	111	0000000	33/7/00
lui	U	0110111			37
addw	R	0111011	000	0000000	3B/0/00
subw	R	0111011	000	0100000	3B/0/20
sllw	R	0111011	001	0000000	3B/1/00
srlw	R	0111011	101	0000000	3B/5/00
sraw	R	0111011	101	0100000	3B/5/20
beq	SB	1100011	000		63/0
bne	SB	1100011	001		63/1
blt	SB	1100011	100		63/4
bge	SB	1100011	101		63/5
bltu	SB	1100011	110		63/6
bgeu	SB	1100011	111		63/7
jalr	I	1100111	000		67/0
jal	UJ	1101111			6F
ecall	I	1110011	000	000000000000	73/0/000
ebreak	I	1110011	000	000000000001	73/0/001

Notes: 1) The Word version only operates on the rightmost 32 bits of a 64-bit registers
2) Operation assumes unsigned integers (instead of 2's complement)
3) The least significant bit of the branch address in jalr is set to 0
4) (signed) Load instructions extend the sign bit of data to fill the 64-bit register
5) Replicates the sign bit to fill in the leftmost bits of the result during right shift
6) Multiply with one operand signed and one unsigned
7) The Single version does a single-precision operation using the rightmost 32 bits of a 64-bit F register
8) Classify writes a 10-bit mask to show which properties are true (e.g., –inf, -0,+0, +inf, denorm, ...)
9) Atomic memory operation; nothing else can interpose itself between the read and the write of the memory location

The immediate field is sign-extended in RISC-V

PSEUDO INSTRUCTIONS ③

MNEMONIC	NAME	DESCRIPTION	USES
beqz	Branch = zero	if(R[rs1]==0) PC=PC+{imm,1b'0}	beq
bnez	Branch ≠ zero	if(R[rs1]!=0) PC=PC+{imm,1b'0}	bne
fabs.s,fabs.d	Absolute Value	F[rd] = (F[rs1]< 0) ? –F[rs1] : F[rs1]	fsgnx
fmv.s,fmv.d	FP Move	F[rd] = F[rs1]	fsgnj
fneg.s,fneg.d	FP negate	F[rd] = –F[rs1]	fsgnjn
j	Jump	PC = {imm,1b'0}	jal
jr	Jump register	PC = R[rs1]	jalr
la	Load address	R[rd] = address	auipc
li	Load imm	R[rd] = imm	addi
mv	Move	R[rd] = R[rs1]	addi
neg	Negate	R[rd] = –R[rs1]	sub
nop	No operation	R[0] = R[0]	addi
not	Not	R[rd] = ~R[rs1]	xori
ret	Return	PC = R[1]	jalr
seqz	Set = zero	R[rd] = (R[rs1]== 0) ? 1 : 0	sltiu
snez	Set ≠ zero	R[rd] = (R[rs1]!= 0) ? 1 : 0	sltu

ARITHMETIC CORE INSTRUCTION SET ②

RV64M Multiply Extension

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	NOTE
mul,mulw	R	MULtiply (Word)	R[rd] = (R[rs1] * R[rs2])(63:0)	1)
mulh	R	MULtiply High	R[rd] = (R[rs1] * R[rs2])(127:64)	
mulhu	R	MULtiply High Unsigned	R[rd] = (R[rs1] * R[rs2])(127:64)	2)
mulhsu	R	MULtiply upper Half Sign/Uns	R[rd] = (R[rs1] * R[rs2])(127:64)	6)
div,divw	R	DIVide (Word)	R[rd] = (R[rs1] / R[rs2])	1)
divu	R	DIVide Unsigned	R[rd] = (R[rs1] / R[rs2])	2)
rem,remw	R	REMainder (Word)	R[rd] = (R[rs1] % R[rs2])	1)
remu,remuw	R	REMainder Unsigned (Word)	R[rd] = (R[rs1] % R[rs2])	1,2)

RV64A Atomtic Extension

amoadd.w,amoadd.d	R	ADD	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] + R[rs2]	9)
amoand.w,amoand.d	R	AND	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] & R[rs2]	9)
amomax.w,amomax.d	R	MAXimum	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amomaxu.w,amomaxu.d	R	MAXimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amomin.w,amomin.d	R	MINimum	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amominu.w,amominu.d	R	MINimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amoor.w,amoor.d	R	OR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] | R[rs2]	9)
amoswap.w,amoswap.d	R	SWAP	R[rd] = M[R[rs1]], M[R[rs1]] = R[rs2]	9)
amoxor.w,amoxor.d	R	XOR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] ^ R[rs2]	9)
lr.w,lr.d	R	Load Reserved	R[rd] = M[R[rs1]], reservation on M[R[rs1]]	
sc.w,sc.d	R	Store Conditional	if reserved, M[R[rs1]] = R[rs2], R[rd] = 0; else R[rd] = 1	

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

REGISTER NAME, USE, CALLING CONVENTION ④

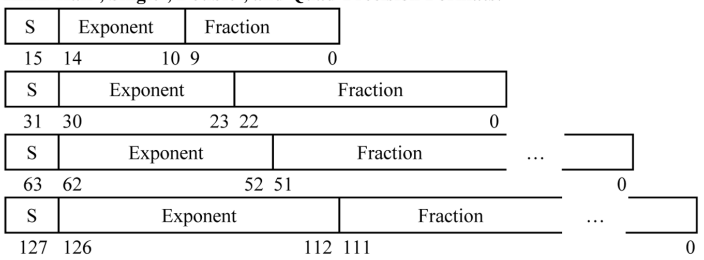
REGISTER	NAME	USE	SAVER
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/Frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/Return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-f7	ft0-ft7	FP Temporaries	Caller
f8-f9	fs0-fs1	FP Saved registers	Callee
f10-f11	fa0-fa1	FP Function arguments/Return values	Caller
f12-f17	fa2-fa7	FP Function arguments	Caller
f18-f27	fs2-fs11	FP Saved registers	Callee
f28-f31	ft8-ft11	R[rd] = R[rs1] + R[rs2]	Caller

IEEE 754 FLOATING-POINT STANDARD

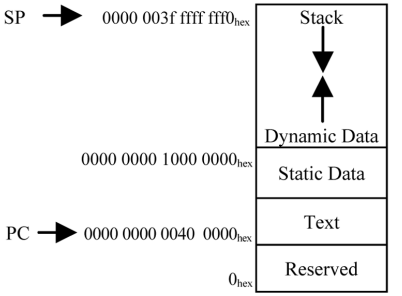
$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

where Half-Precision Bias = 15, Single-Precision Bias = 127,
Double-Precision Bias = 1023, Quad-Precision Bias = 16383

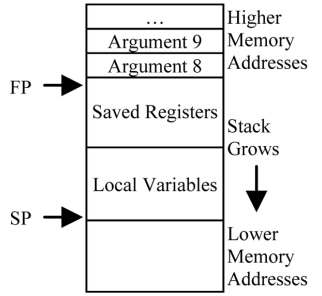
IEEE Half-, Single-, Double-, and Quad-Precision Formats:



MEMORY ALLOCATION

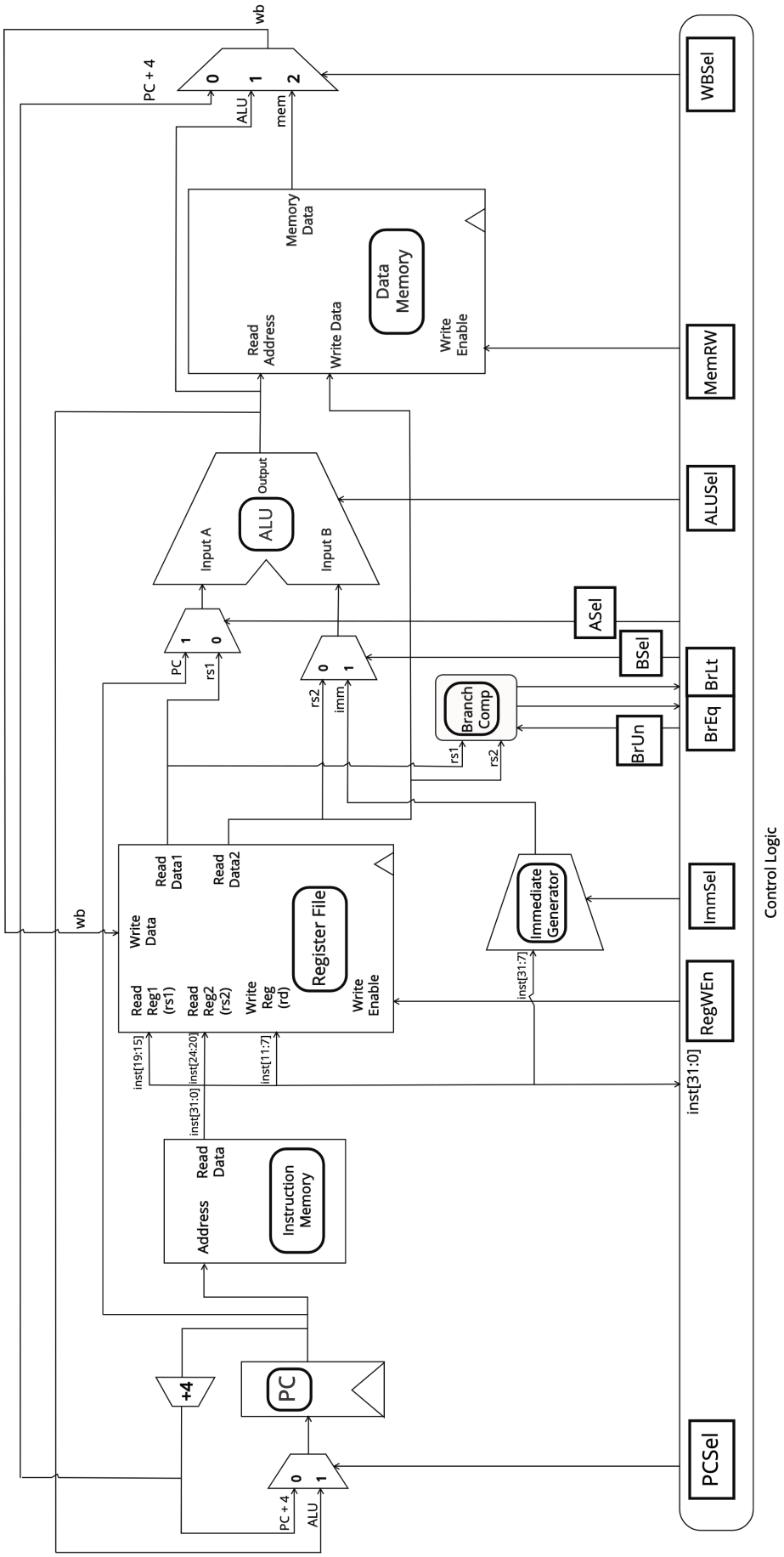


STACK FRAME



SIZE PREFIXES AND SYMBOLS

SIZE	PREFIX	SYMBOL	SIZE	PREFIX	SYMBOL
10^3	Kilo-	K	2^{10}	Kibi-	Ki
10^6	Mega-	M	2^{20}	Mebi-	Mi
10^9	Giga-	G	2^{30}	Gibi-	Gi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti
10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi
10^{-3}	milli-	m	10^{-15}	femto-	f
10^{-6}	micro-	μ	10^{-18}	atto-	a
10^{-9}	nano-	n	10^{-21}	zepto-	z
10^{-12}	pico-	p	10^{-24}	yocto-	y


wb
PC + 4
ALU
mem
0
1
2
Memory Data
Data Memory
Read Address
Write Data
Write Enable
Input A
Input B
Output
ALU
PC
rs1
rs2
imm
Branch Comp
rs1
rs2
Read Data1
Read Data2
Write Data
Register File
Read Reg1 (rs1)
Read Reg2 (rs2)
Write Reg (rd)
Write Enable
Immediate Generator
inst[31:7]
inst[19:15]
inst[24:20]
inst[11:7]
inst[31:0]
Address
Read Data
Instruction Memory
+4
PC
PC + 4
ALU
WBSel
MemRW
ALUSel
ASel
BSel
BrLt
BrEq
BrUn
ImmSel
RegWEn
PCSel
inst[31:0]
Control Logic

1	8	23
Sign	Exponent	Mantissa/Significand/Fraction

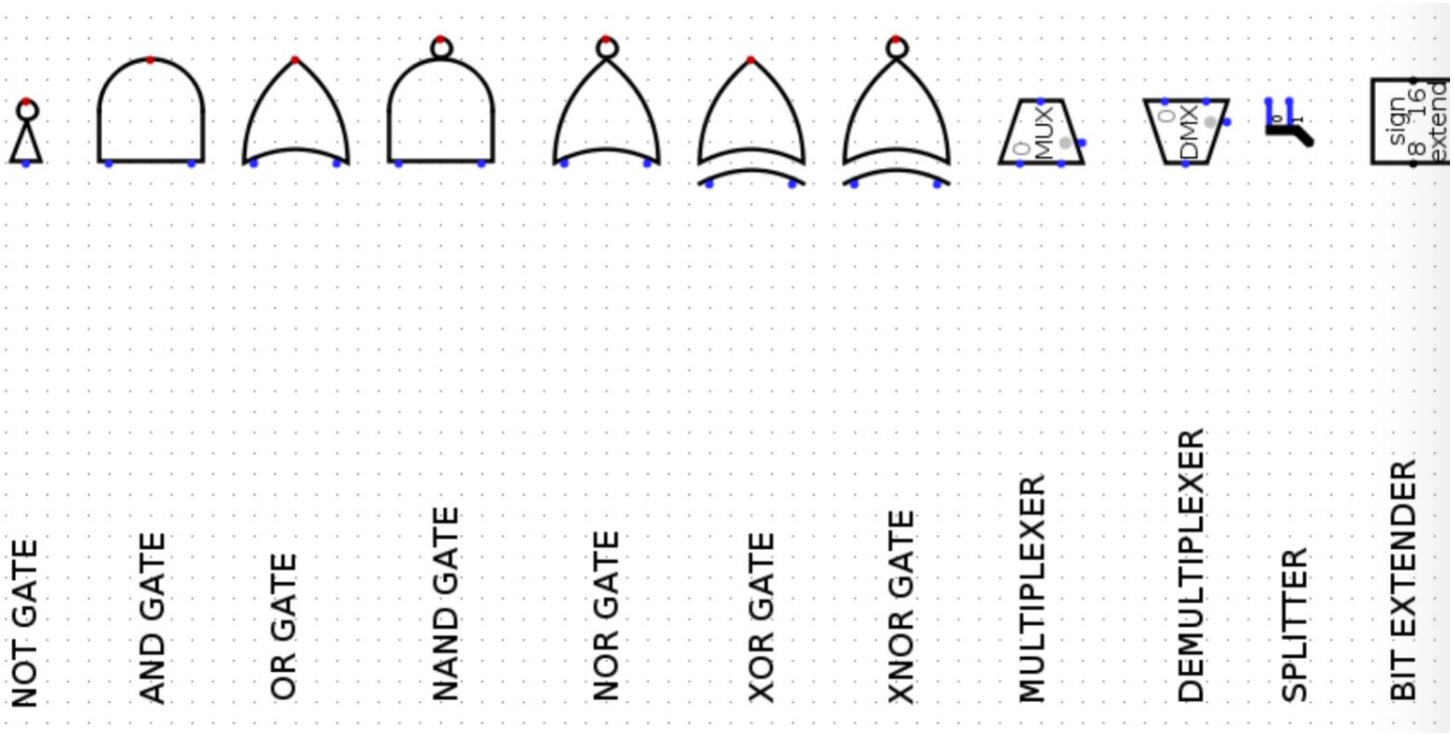
For normalized floats:

$$\textbf{Value} = (-1)^{Sign} * 2^{Exp-Bias} * 1.\textbf{significand}_2$$

For denormalized floats:

$$\textbf{Value} = (-1)^{Sign} * 2^{Exp-Bias+1} * 0.\textbf{significand}_2$$

Exponent	Significand	Meaning
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN



Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▓	209	D1	╤	241	F1	±
146	92	Æ	178	B2	│	210	D2	╥	242	F2	≥
147	93	ô	179	B3	┤	211	D3	╙	243	F3	≤
148	94	ö	180	B4	╡	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╢	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╖	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╕	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╣	216	D8	╪	248	F8	°
153	99	Ö	185	B9	║	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	╗	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╝	219	DB	█	251	FB	√
156	9C	£	188	BC	╜	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╛	221	DD	▌	253	FD	²
158	9E	Pts	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Use this sheet as scratch paper

Use this sheet as scratch paper

Use this sheet as scratch paper