HONOR CODE

- I have not used any online resources during the exam.
- I have not obtained any help either from anyone in the class or outside when completing this exam.
- No sharing of notes/slides/textbook between students.
- NO SMARTPHONES.
- **CANVAS ANSWERS WILL BE LOCKED AFTER 1ST TRY.**

Questions Sheet.

Read all of the following information before starting the exam:

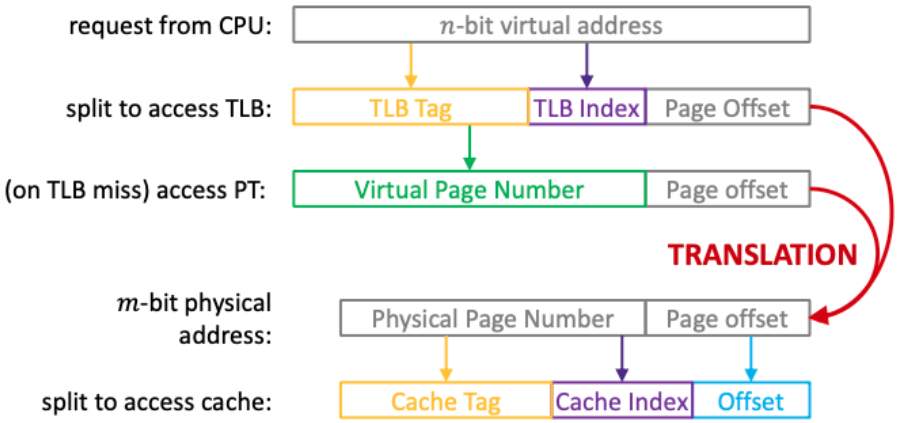
- For each question fill out the appropriate choice or write text on Canvas page. Also type clearly on in the exam on the appropriate text.
- IF THE MULTIPLE CHOICE ANSWER IS WRONG WE WILL MARK THE ANSWER WRONG. IF THE MULTIPLE-CHOICE ANSWER IS CORRECT, WE WILL READ THE WRITTEN PORTION.
- Show all work, clearly and in order, if you want to get full credit.
- I reserve the right to take off points if I cannot see how you logically got to the answer (even if your final answeris correct).
- Circle or otherwise indicate your final answers.
- Please keep your written answers brief; be clear and to the point.
- I will take points off for rambling and for incorrect or irrelevant statements. This test has six problems.

Section Virtual Memory 22 points. Canvas Q1-Q22

```
Refer slide deck L21-VM-III Week 8 if you need to.
```

The chart below shows how memory accesses are treated in a system. The table below describes the parameters int he memory system. Please use the data below to answer question groups Q1,Q2,Q3,Q4 on canvas.

```
CAUTION: When converting from binary to hex you can always pad the MSB
e.g., 10 1010 (6 bit field) in hex is 0010 1010 (2 0s padded in MSB)
is 0x2a .
```



Parameter	Value
Physical address bits	18
Size of page	1KB or 1024 bytes
Virtual address bits	22
----------	----------
TLB Sets	4
TLB Ways	4
TLB Size	16 entries
----------	----------
Cache block	16 bytes
Cache size	256 bytes
Cache Sets	4
Cache Ways	4

Terminology

- VPN - Virtual page number
- Index (Set index of cache or TLB)
- PPN - Physical page number
- INVALID. TLB entry is invalid
- TLB-T (TLB Tag)

TLB

Way 0	PPN
TLB-T:[0xe8] Index:[0x0]	INVALID
TLB-T:[0x2fa] Index:[0x1]	0x8d
TLB-T:[0x71] Index:[0x2]	INVALID
TLB-T:[0x2ca] Index:[0x3]	0x70

Way 1	PPN
TLB-T:[0x23c] Index:[0x0]	0x91
TLB-T:[0x2fc] Index:[0x1]	0xfa
TLB-T:[0x5] Index:[0x2]	0x99
TLB-T:[0x2db] Index:[0x3]	0x13

Way 2	PPN
TLB-T:[0x1ce] Index:[0x0]	INVALID
TLB-T:[0x301] Index:[0x1]	0xbd
TLB-T:[0x236] Index:[0x2]	INVALID
TLB-T:[0x21a] Index:[0x3]	INVALID

Way 3	PPN
TLB-T:[0x118] Index:[0x0]	INVALID
TLB-T:[0xf] Index:[0x1]	0x33
TLB-T:[0x298] Index:[0x2]	INVALID
TLB-T:[0x29d] Index:[0x3]	0x1f

Page Table (Partial)

```
CAUTION: Only partial table relevant to the questions are shown.
```

VPN	PPN	Valid
0xb2b	0x70	1
0x8e9	0x8d	1
0xc05	0xbd	1
0x2db	0x13	1
0x738	----	0
0x3a0	----	0
0xbe9	0x9d	1
0x1c6	----	0
0x8f0	0x91	1
0xbf1	0x47	1
0x016	0x99	1

Cache

- Way 0

Way 0	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Tag: [0x917] Index: [0x0]	0xe8	0x13	0x9e	0x26	0xaf	0xc5	0x72	0x44	0xbc	0x6d	0x78	0x50	0x66	0x2f	0x66	0x8f
Tag: [0x8d5] Index: [0x1]	0xe4	0x2b	0x0d	0xd3	0xa0	0xb2	0x0f	0x9a	0xe9	0x7e	0xc8	0x0e	0x1e	0x13	0xea	0x6a
Tag: [0x707] Index: [0x2]	0x5a	0xd9	0xc9	0x38	0x50	0xba	0x35	0x0c	0x4c	0x8c	0xd7	0xc7	0xaa	0x79	0x2f	0x0d
Tag: [0x7b7] Index: [0x3]	0x57	0xb4	0x4c	0xda	0x4a	0xbb	0xc6	0x25	0x8c	0x5f	0x7a	0x24	0xd5	0xac	0xc4	0xc3

- Way 1

Way 1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Tag: [0x133] Index: [0x0]	0x4f	0xa0	0x34	0x03	0x7c	0x72	0x20	0x46	0x12	0xbd	0x7b	0x74	0xbe	0xf7	0x38	0x11
Tag: [0x761] Index: [0x1]	0xb9	0x0f	0x68	0x06	0xe4	0xb7	0xad	0x7d	0xca	0xb1	0x83	0x10	0xa2	0x9e	0x9f	0xd8
Tag: [0x336] Index: [0x2]	0x27	0x90	0x08	0x04	0x50	0xbe	0xd8	0x7b	0x92	0x08	0x9b	0xb7	0x6d	0xe1	0xc2	0x2e
Tag: [0xbaf] Index: [0x3]	0xcb	0x7d	0x7e	0x48	0x04	0x40	0xba	0x33	0x79	0xca	0x50	0x1d	0x4f	0xf5	0xbd	0x8e

- Way 2

Way 2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Tag: [0x39] Index: [0x0]	0xe6	0x03	0x8b	0x4f	0xcc	0x42	0x16	0xa7	0xd0	0x8d	0x9b	0x7d	0x9e	0x10	0x36	0x9d
Tag: [0xdc0] Index: [0x1]	0x47	0x8f	0x7a	0x8f	0x70	0x57	0xbd	0x90	0xef	0xec	0x5f	0xb4	0x1e	0x62	0xe8	0xd6
Tag: [0x1f8] Index: [0x2]	0xce	0xbd	0xa3	0xd5	0x22	0x46	0xb9	0x27	0xee	0x57	0x28	0xe8	0x7a	0x27	0x2f	0x3c
Tag: [0xfab] Index: [0x3]	0xee	0xef	0xe6	0xcd	0x00	0xe7	0x3f	0xd9	0x65	0xd0	0xcc	0x60	0x27	0x80	0x7b	0xe3

- Way 3

Way 3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Tag: [0x2b0] Index: [0x0]	0x35	0x76	0x31	0xa3	0x23	0x54	0x74	0x1e	0xc1	0x16	0xd3	0x18	0x59	0xfb	0xdf	0x2a
Tag: [0xbdd] Index: [0x1]	0xc5	0x87	0x52	0x27	0x94	0xcd	0xe4	0x53	0xeb	0xb5	0xa2	0xd9	0x28	0x61	0x34	0x43
Tag: [0x47c] Index: [0x2]	0xbe	0xd0	0xfa	0xd1	0xad	0x91	0xb4	0xb4	0x2c	0x43	0xce	0x45	0xc0	0xdb	0x73	0x44
Tag: [0x815] Index: [0x3]	0x27	0x3f	0xce	0xd8	0xc7	0x1d	0xbe	0x2c	0xa5	0x4f	0x0d	0x13	0x55	0xde	0xa5	0xed

Common questions. Canvas Q1-Q2

1. How many bits is the VPN ?

- 12

2. How many bits is the PPN ?

- 8

For the virtual address 0x2cade0 answer the following Canvas Q3-Q12

What is the VPN

- 0xb2b

What is the TLB tag.

- 0x2ca
- Is it a TLB hit or miss
  - Hit
- Is it a page fault
  - NO
- What is the PPN ?
  - 0x70
- what is the cache tag ?
  - 0x707
- what is the cache index
  - 0x2
- What is the byte offset
  - 0x0
- Is it a cache hit or miss
  - Hit
- What is the data byte
  - 0x5a

For the virtual address 0x301754 answer the following. Canvas Q13-Q22

- What is the VPN
  - 0xc05
- What is the TLB tag.
  - 0x301
- Is it a TLB hit or miss
  - Hit
- Is it a page fault
  - NO
- What is the PPN ?
  - 0xbd
- what is the cache tag ?
  - 0xbdd
- what is the cache index
  - 0x1
- What is the byte offset
  - 0x4
- Is it a cache hit or miss
  - Hit
- What is the data byte
  - 0x94 (Correct). If you answered None of the above or 0x27 you got points.

B. Section Cache I Questions. 15 points. Canvas Q23-Q25

```
Refer L14-Cache I if you have to
```

Let the A[0] be at 0x00000 and B[0] be at 0x10000. The size of an integer is 4 bytes. Size of each array is 1024 ints. Describe the behavior of the following code when run on the cache and answer the questions. Assume that there is 1 level of cache and it is completely empty when starting this program. The size of the cache is 2 KB, 16 sets, 16 ways and 8 byte blocks.

```
int A[1024], B[1024];
void loops() {
  // Loop 1
  for (int index = 0; index < 32; index++) {
    B[index] = 0xff;
    A[index] = 0xff;
  }
  // Loop 2
  for (int index = 32; index < 1024 index++) {
    B[index] = B[index - 16] + A[index -  16] ;
    A[index] = B[index -  8] + A[index -  8] ;
  }
}
1 level of cache
+--------------+
|  16 sets     |
|  16 ways     |
|  8 byte block|
+--------------+

```

23. What is the miss rate for loop 1? (Assume that only loop 1 runs). 5 points

1/2

Miss/Hit Pattern is MMHH.The hit rate is 50%.

24. What is in the cache at the end of loop 1? 5 points

A[0:31], B[0:31]

25. What is the miss rate for loop 2 ? Assume that loop 1 has already run to completion and has warmed up the cache. 5 points

Hit rate: 10/12
Miss rate: 2/12 or 1/6

The miss/hit pattern is H A (index - 8) H A(index-16) M (A-index) H HM (iteration=0), HHHHHH (iteration 1).
Each 6 accesses are from one iteration of the for loop.
10/12 accesses

In the first iteration of loop 2, B[index-16], A[index-16], B[index-8] and A[index-8] are already in the cache (since loop 1 has run, see answer to question 24). (index=32 so B[16], A[16], B[24], A[24] are all in the cache)
B[32] and A[32] are not in the cache. So in this iteration you have 4 hits and 2 misses.
In the second iteration, since each block has 8 bytes (2 ints), B[33] and A[33] are now in the cache since they were brought in during the first loop iteration when you accessed B[32] and A[32]. So the second iteration has 6 hits.
So after two iterations you have 10 hits and 2 misses.
This pattern repeats for the 3rd and 4th iterations (miss on A[34] and B[34] and hit on all the others), and continues to repeat for all other pairs of iterations in loop 2.

extra: How does the 16 ways work, what elements are stored in each way?
Each way stores a different block that belongs to the same cache set. All ways in a set have the same set index bits but different tag bits. Please review the cache lecture (week 7) for more details about set-associative caches.

C. Section Cache II Questions. 20 points. Canvas Q26-Q31

```
int src[2048]; Address - 0x0000
int dest[2048]; Address - 0x1000
for (int i = 0; i<2048; i += 4) {
  b[i] = a[i];
}

Cache Parameters

sizeof(int) - 4 bytes
1 level of cache
+------------------------+
|  512 byte              |
|Direct mapped           |
|32 bytes/block or 8 ints|
+------------------------+

Cache layout

    1 way (Direct mapped)
    +------------------+
    |(32 byte or 8 int)|
    +------------------+
    |                  |
    +------------------+
16  |                  |
sets+------------------+
```

26. Assuming the total size of the physical address is 32 bits. What is the number of bits required by tag, index and offset (4 points)

Tag: 23 Index: 4 Offset: 5

27. What is the hit rate of this direct-mapped cache? (4 points)

0 . B[i] conflicts with A[i]. Direct mapped cache. A is brought in and then B is brought in and evicts A. Nothing hits.

28. What type of misses occur (Conflict, Compulsory, Capacity) ? (2 points)

Conflict and Compulsory.

29. Now consider a 2-way set associative cache. 512 bytes. 8 words/block.
    What is the hit rate ? (4 points)

1/2. MMHH
A[0] and A[3] fall in the same cache line
B[0] and B[3] fall in the same cache line . i+=4 means you access every fourth element. A block has 8 words and you access 2.30.
What type of misses occur (Conflict, Compulsory, Capacity) ? (2 points)

Compulsory

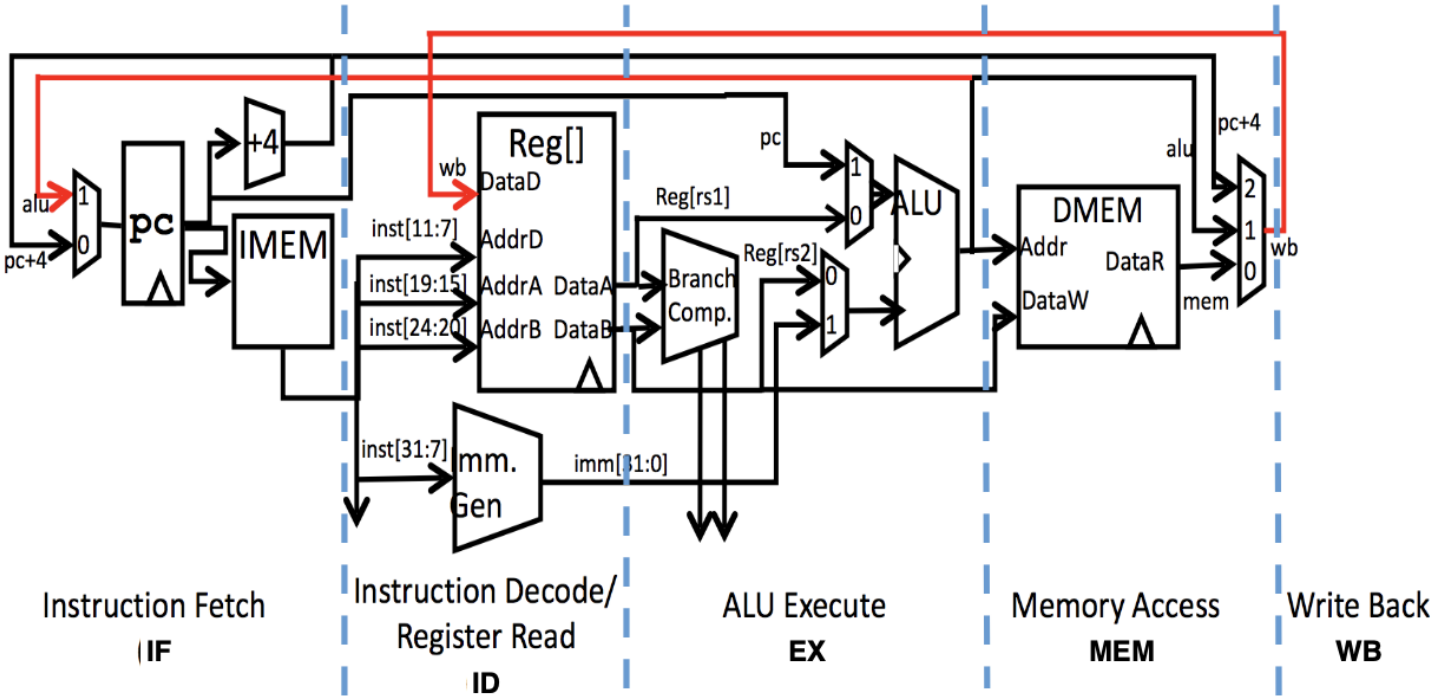
31. Now consider a 4-way set associative cache. 512 bytes. 8 words/block.
    What is the hit rate ? (4 points)

1/2

D. RISC-V Pipeline 20 points. Canvas Q32-Q41

`Refer slide deck L29-Hazard Week 11 if you need to.`

Consider a typical 5-stage (Fetch,Decode,EXecute,Memory,WriteBack) pipeline.Assume pipeline registers exist where the dotted lines are



This pipeline is more simple than the one you dealt with in the assignment.

- Forwarding/Bypassing is not implemented; dependent instructions will have to wait in the ID stage.
  (`CAUTION: In lecture we illustrated dependent instructions waiting in the IF stage`)
- Following a branch, the next instructions always fetches from PC+4 until the branch is resolved in the WB stage
  (`CAUTION: Note that the lecture slides resolved branch in the EX stage`). Flush the pipeline if branch is taken.
- We can read and write from the same registers or memory location in the same clock cycle. Any memory location can be accessed

Answer questions based on the following program

```
1  addi x9, x0, 0xF    # Instruction 1
2  addi x18, x0, 0     # Instruction 2
3  beq x9, x18, exit   # Instruction 3
4  lw x9, 10(x8)       # Instruction 4
5  xor x9, x9, x18     # Instruction 5
6  exit:
7    sw x9, 10(x8)     # Instruction 6
```

`Hint: Start by creating a pipeline sheet similar to Assignment 6 (with pen and paper)`

Ins-Cycle	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
addi x9, x0, 0xF	ID	EX	MEM	WB											
addi x18, x0, 0	IF	ID	EX	MEM	WB										
beq x9, x18, exit		IF	ID	ID	ID	EX	MEM	WB							
lw x9, 10(x8)			IF	IF	IF	ID	EX	MEM	WB						
xor x9, x9, x18						IF	ID	ID	ID	EX	MEM	WB			
sw x9, 10(x8)									IF	ID	ID	ID	EX	MEM	WB

32. In which cycle does `addi x18,x0,0` (instruction 2) run the EX stage ?

Cycle 4.

33. In which cycle does `beq x9,x18,exit` (instruction 3) read the registers?

Cycle 6. beq reads register x18 and instruction 2 writes register x18.

34. In which cycle does the `lw x9, 10(x8)` (instruction 4) start the IF stage ?

Cycle 4 or Cycle 6 depending on whether beq stalled in IF or ID.

35. In which cycle does the `lw x9, 10(x8)` read the registers ?

Cycle 7.

36. In which cycle does the `xor x9, x9, x18` (instruction 5) reach the IF stage ?Cycle 7.

37. In which cycle does the `xor x9,x9,x18` (instruction 5) read the registers ?

Cycle 10. Not until lw reaches the WB stage. xor is dependent on the load.
load writes x9. xor reads x9

38. In which stage is `sw x9, 10(x8)` stalled and how many cycles?

ID and spends 3 cycles in it (2 cycles of which are stalls).

Now when comparing to baseline pipeline (forwarding/bypassing implemented, branch resolves in EX etc). Then sw would start at IF 8 (but here it starts at 10 so 2 additional cycles). Then further 2 cycles for ID (so total 4 cycles).

39. In which cycle does the `sw x9, 10(x8)` (instruction 6) write the memory location ?Cycle 15

40. How many instructions are stalled due to data hazards ?

3

41. How many cycles do we have stall in total for this program ? i.e., Consider a program with 6 instruction and no hazards and ran to completion in T cycles. This program completed in T_hazard cycles. What is (T_hazard - T)?

6

The first thing to notice for this question is that the datapath does not implement bypassing. Recall that instructions read their registers in stage ID, and write registers in WB. These restrictions mean that if instruction B needs a register that instruction A writes, then B can start the ID stage in the same cycle.

Instruction 2: This doesn't have any data dependencies, so we just need to worry about structural hazards. It can start as soon as the IF stage is available (cycle 2.

Instruction 3: This instruction reads register x18 which was written by the previous instruction. Therefore we must wait until the previous has reached its WB stage before running beq's ID stage (c6).

Instruction 4: lw s1 0xc(s0):At this point, the result of the branch doesn't matter because it is always predicted to be not taken. Also, the branch doesn't write any registers,so we don't have any data dependencies and can start as soon as the stages are available. In this case, the fetch can start on c4, but the decode has to wait until beq is in the EX stage. (Cycle 7)

Instruction 5: This instruction cannot start until the load instruction is in the ID stage.

Instruction 6: Data hazard on instruction 5. Store reads register x9, xor calculates register x9. Branch is not taken so all instructions can continue running.

IF branch is taken, then we have to wait till WB to flush (following instructions may enter the pipeline)

Ins-Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
addi x9, x0, 0xF	IF	ID	EX	MEM	WB									
addi x18, x0, 0xF		IF	ID	EX	MEM	WB								
beq x9, x18, exit			IF	ID	ID	ID	EX	MEM	WB					
lw x9, 10(x8)				IF	IF	IF	ID	EX	EX					
xor x9, x9, x18							IF	ID	ID					
sw x9, 10(x8)								IF	IF	IF	ID	EX	MEM	WB

E. RISC-V Datapath 20 points. Canvas Q42-Q51

We wish to introduce a new instruction into our RISC-V datapath.
`RELU` . This is related to the relu operation in assignment 3. The instruction works as follows.

```
1 # rd_rs2, is a register that acts as a source
2 # and destination register
3 RELU rd_rs2, offset(rs1)
4 if (0<=R[rd_rs2])
5   MEM[R[rs1]+offset] = R[rd_rs2]
6 else
7   MEM[RS[rs1]+offset] = 0
8   R[rd_rs2] = 0
```

It combines the semantics of branch, load and store.

- Like a load it performs arithmetic using the ALU for calculating `R[rs1]+offset` the memory address to be modified.
- Like a store it updates the MEM[address] with a value (either rd or 0).
- Like a branch it performs comparison. However the operands used are different. In a typical branch operation A<=B A is obtained from rs1 and B is obtained from rd_rs2. In RELU, A is always 0 and B is rd_rs2.
- Typically, the branch comparison will modify PC. However, here the branch comparison influences what value is stored to Memory, either R[rd_rs2] or 0.
- Further, the branch comparison influences the value of rd in a load operation, if the branch comparison fails the R[rd_rs2] is 0.

```
Caution 1: In a typical RISC-V instruction rs2 field is used as source only and rd as destination only.
In this case we are using the rd field also as a source when performing the comparison operation line 4
and writing to memory (line 5). We are also using rd field as a destination register in line 8.
```

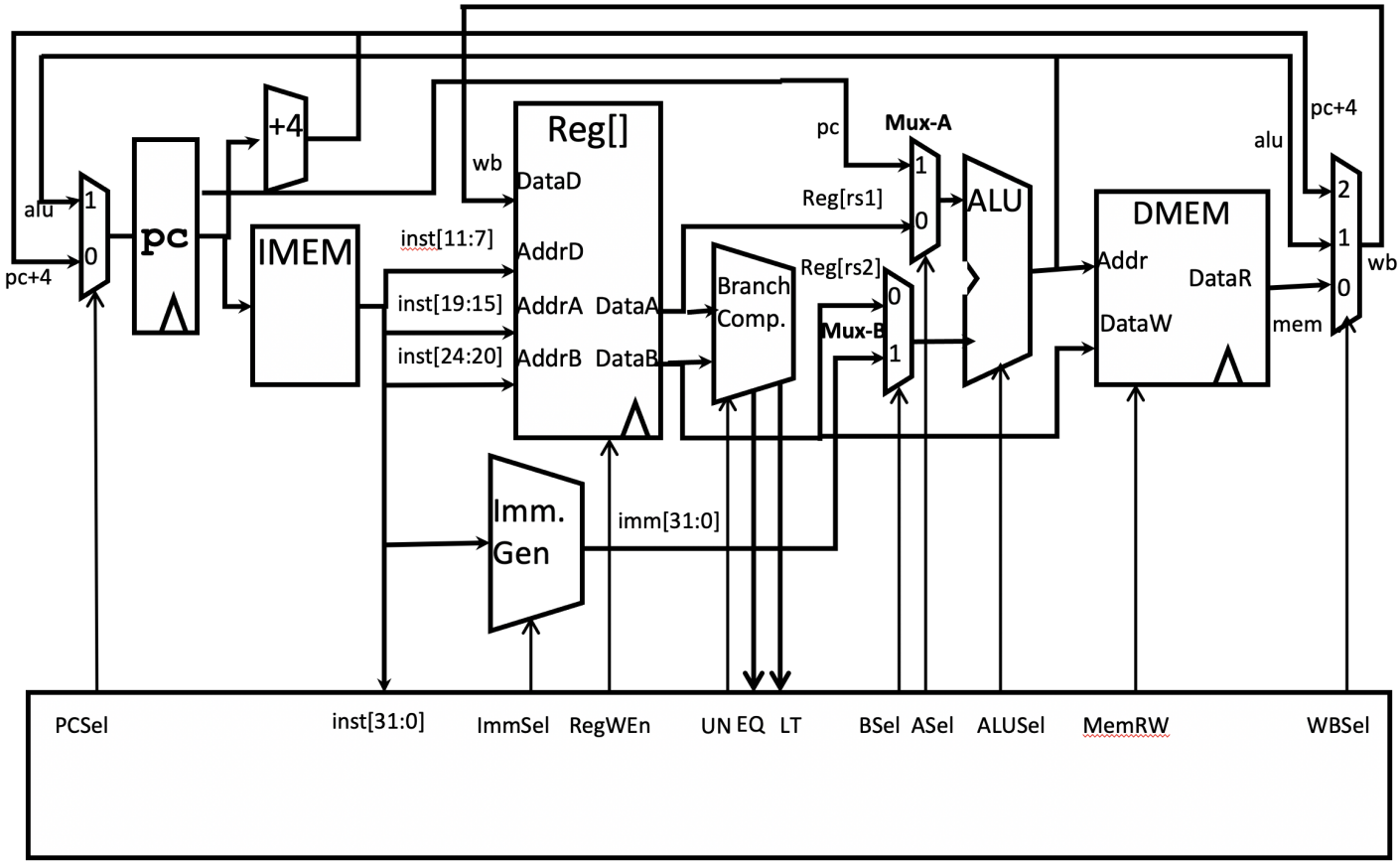
Given the single cycle datapath below, select the correct modifications in parts such that the datapath executes correctly for this new instruction (and all other instructions!). You can make the following assumptions:

- We have a new control signal `RELU` which is 1 if the instruction being decoded is a RELU
- ALUsel is add when we have a RELU instruction
- The immediate generate sign extends the offset similar to load instructions.

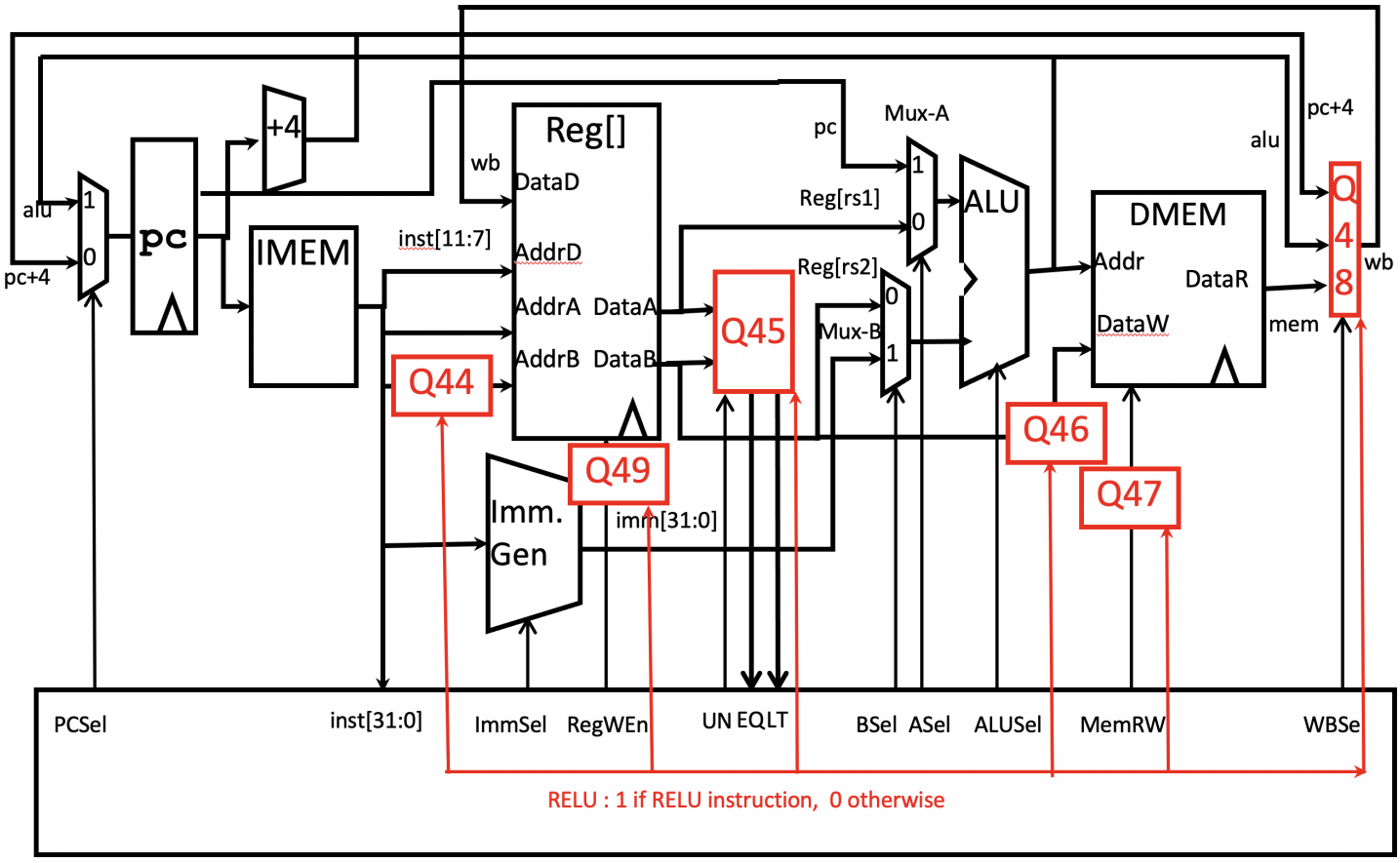
```
Caution 2: Pay careful attention to which input line is 1 and which line is 0 in the muxes.
Some muxes choose top-most input as 0, some choose bottom-most input as 0
```

```
Hint: YOU DO NOT REQUIRE TRUTH TABLES
Try writing down in plain english or reading out the logic
to yourself e.g, !(A<=B) is A is not equal to B and A is not LT (less than) B
```

Baseline Pipeline



Pipeline with RELU (Red boxes indicate questions)



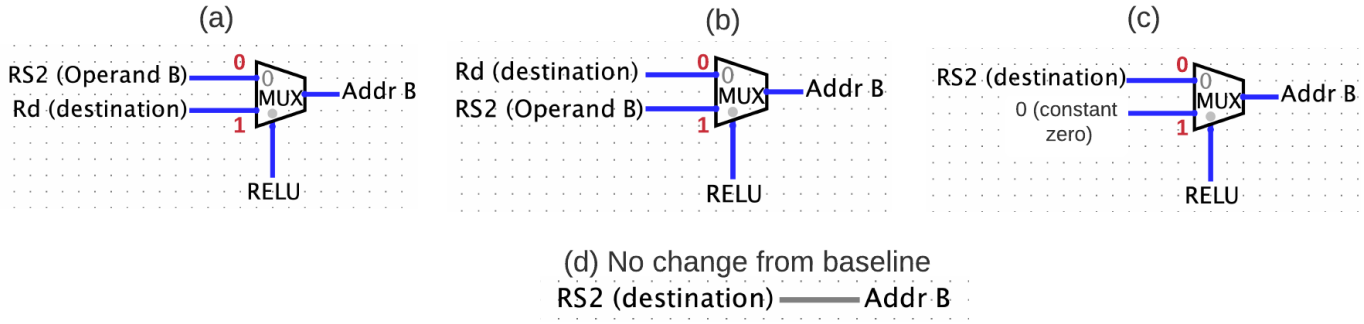
42. What type of instruction is `RELU` ?

I-type Load. 1 source, 1 destinaton, 1 immediate. However the instruction also uses rd as a rs2 for branch comparisons and writing to memory. We use rd as rs2 since rs2 is the only register that is forwarded to memory as well in stores.

43. Which instruction field can be written to memory in the baseline pipeline?

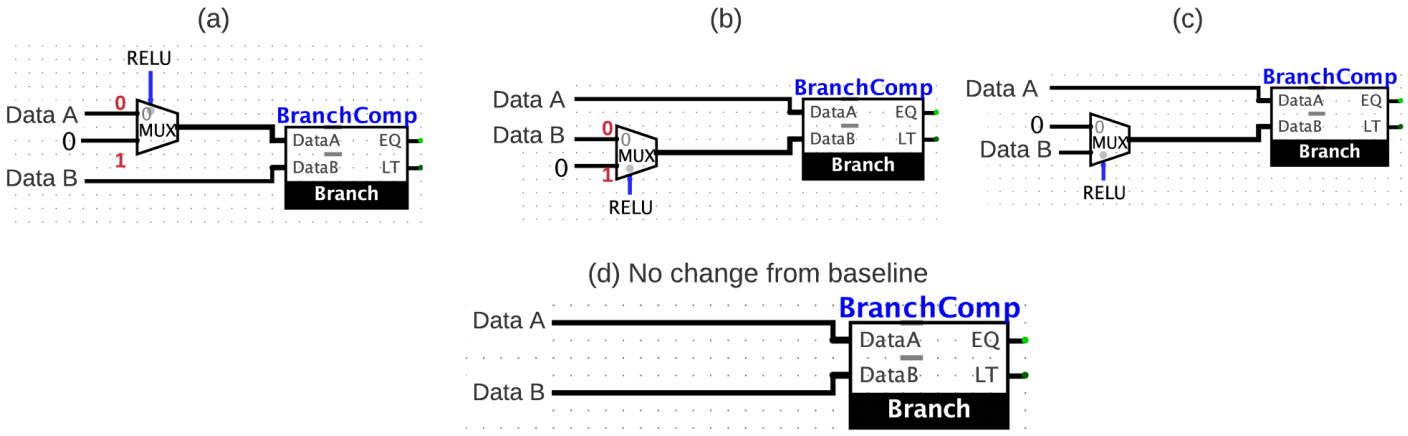
rs2

44. Consider the following modifications to the source `Reg[]` inputs. Which configuration will allow this instruction to execute correctly without breaking the ex-ecution of other instructions in our instruction set?



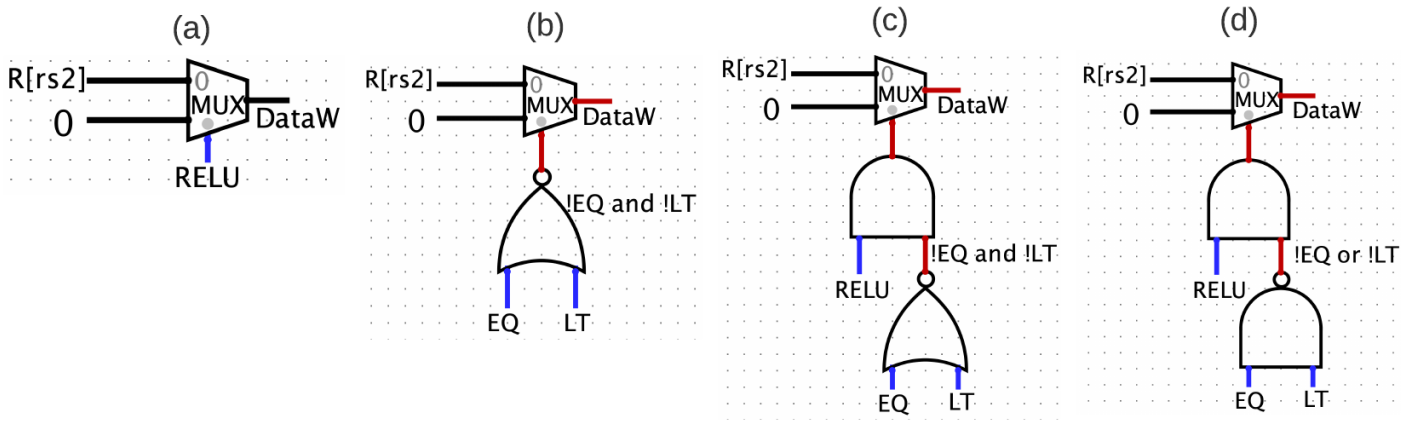
A.

45. Consider the following modifications to the `Branch` . Which con-figuration will allow this instruction to execute correctly without breaking the ex-ecution of other instructions in our instruction set? Branch calculates `A==B` and `A<B`



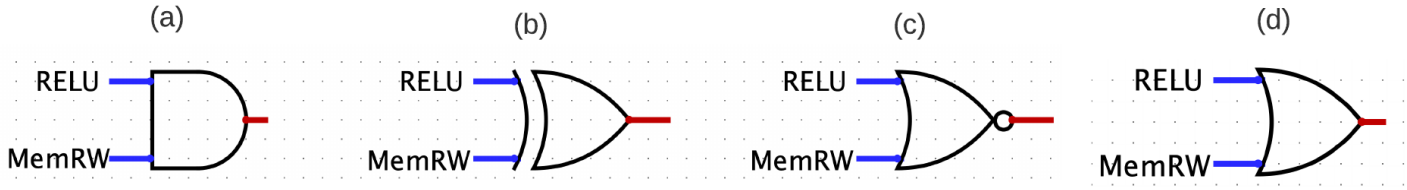
A. All branches are comparisons of type A==B? or A<B? Here if RELU we set A = 0 and B = R[Rs2] (default behavior). So we are comparing 0 < R[rs2]? , 0 == R[Rs2]. If we say NOT (== OR <) i.e, Not (0<R[rs2]) AND NOT (0<R[Rs2]) . Then it is strictly greater 0 >R[rs2]. The else part of the block

46. Consider the following modifications to the `DMEM` inputs. Which configuration will allow this instruction to execute correctly without breaking the execution of other instructions in our instruction set?



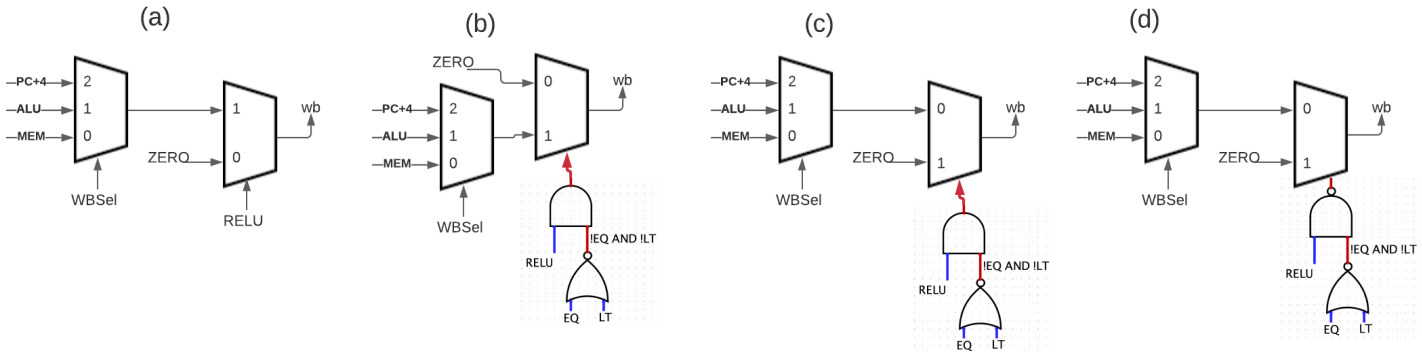
C. If RELU and 0 > R[rs2] then pick 0 as the value to be written to memory, otherwise then block pick R[rs2]. If not RELU also pick R[rs2] e.g., for store instructions

47. Consider the following modifications to the `DMEM` control signal. Which configuration will allow this instruction to execute correctly without breaking the execution of other instructions in our instruction set?



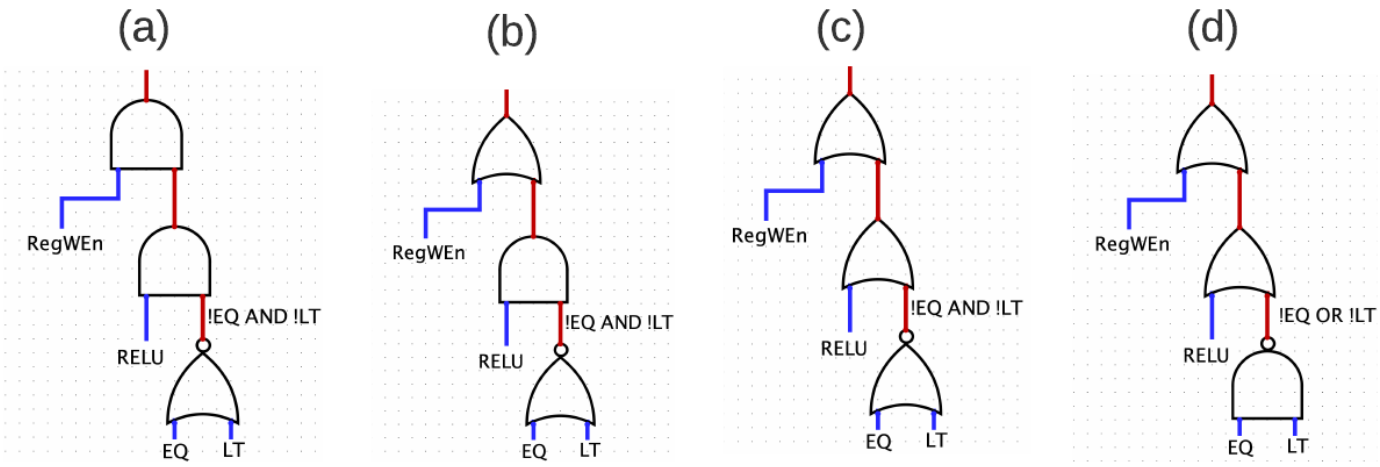
D.

48. Consider the following modifications to the `WB` . Which configuration will allow this instruction to execute correctly



C. If RELU and 0 > R[rs2] then pick 0 as the value to be written to register (the else part of the RELU instruction). If 0<= R[rs2] then does not matter what we pick cause we are going to be disabling RegWen (see next question). If not RELU, pick default.

49. Consider the following modifications to the `RegWEn` mux inputs. Which configuration will allow this instruction to execute correctly



B. If RELU and 0 > R[rs2] then pick 0 as the value to be written to register (the else part of the RELU instruction). If 0<= R[rs2] then does not matter what we pick cause we are going to be disabling RegWen (see next question). If not RELU, pick default.

50. What is the value of ASel?0 (we are using the ALU to calculate rs1+offset. The address for memory)

51. What is the value of BSel?1

F. RISC-V Program 10 points. Canvas Q52-Q53

We will be introducing a new instruction called `lwa` in RISC-V. In baseline RISCV the loads calculate addresses using an immediate and register. However, in many programs typically the address is calculated using 2 registers. The semantics of the `lwa` instruction (lwa rd,rs1,rs2) are dst = MEM[rs1+rs2] e.g., lets say a0=0x4 a1=0x1000 lwa a2,a1,a0. a2 = MEM[0x1004]

You want to impress your friend, so you predict the result of executing the program as it is written, just by looking at it. If the program is guaranteed to execute without crashing, describe what it prints, otherwise explain the bug that may cause a crash.

```
1  .globl main
2  .data
3  a: .string "skayaks"
4  table: .string "ZYXWVUTSRQPONMLKJIHGFEDCBA12345"
5  init: .string "XXXXXXX" # 7 Xs
6
7  .text
8  loop_header:
9          lbu  s2, 0(a0) # Read character ch
10         beqz s2, end
11         addi s7,a0,0
12   addi a0,a0,1
13         la s4,table
14 loop:
15   addi s1, s2, -97 #
16   andi s2,s2,0x1F # andi performs bitwise &  lwa
17   s2,s2,s4 # New instruction
18   sb   s2, 0(a1)
19   addi a1,a1,1
20   j loop_header
21
22 end:
23          ret
24
25 main:
26    la a0, a
27    la a1,init
28    jal loop_header
29    li a0,10
30    ecall
```

52. What 8 memory locations are modified. 5 points

0x10000028-0x1000002F
skayaks\0 - 8 characters
ZYXWVUTSRQPONMLKJIHGFEDCBA12345\0 - 32 characters
The init starts at byte 40. or 0x28 from the base address of the data segment
0x10000000. `a` will not be modified as branch will jump back to loop header.

53. What is the value in those memory locations. 5 points

GOYAYOG

We simply take the bottom 5 bits of the byte value of each letter and then use it as an idex to lookup the table. e.g., the bottom 5 bits of S is 19. Using table as an array of 31 characters. Position number 19 is G.

You need to trace the value of each register in this code as it runs.
a0 starts with the address of string a 'skayaks' and is incremented by 1 inside the loop (line 12)
a1 starts with the address of string init (7 Xs) and is incremented by 1 inside the loop (line 19)

Each loop iteration: You load a character into s2 from string a at a time (read s then k then a etc.) (line 9) and exit if you read the end of string character (line 10) s4 stores the address of string table (line 13)
subtract 97 (ascii for a) from s2, so now s1 contains the order of the letter in the alphabet (line 15)
extract the last five bits from the character in s2 (line 16)
Use s2 as an index into table to get the character in position s2 (line 17)
You store that character into the corresponding character in string init (line 18)
So you basically copy the letters from table into init, where the index of the leter you write is the same as the order of the corresponding letter of string a in the alphabet.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL