Supervision 1

1. Processes & Scheduling

1.

(a)

i. Cache memory is small. It is located physically near the procesor to allow for very fast access times.

ii. Main memory is quite large (on the order of gigabytes in modern computers) and has moderately fast access speeds – slower than accessing a register but faster than writing to a hard drive. Main memory is where program instructions and also most program data are stored.

iii. Registers are very small, usually only storing a single value, and have very fast access times. They are used to store intermediate values during calculations and to store the few values which are currently being used by the processor. * ✓

* and some other values the processor needs (e.g. the address of the current instruction)

(b) When the main memory is getting full, the OS can allocate part of the disc as virtual memory which can be used instead. It is much slower to access than the main memory but it increases the overall amount of memory available. ✓

2.

(a)

i. (Typically) 4 bytes totalling 32 bits in which the presence of a set bit in position n signals that 2^n should be added to form the integer. ✓

ii. Positive integers are represented as above. For negative integers, either the first bit represents whether the integer should be interpreted as negative, or, in two's complement, all of the bits of the corresponding positive integer are flipped, and 1 is added (discarding overflow) ✓ in practice: all modern CPUs use 2's complement

iii. An array of characters, one after the other in continuous memory. A character in ASCII is one 7 bits, the first 2 bits of which represent which keyboard is being used (e.g. 10 for uppercase, 11 for lowercase) and the remaining 5 bits are the position in the alphabet (e.g. 00001 for A, 00010 for B). In other encodings each character can take up more than 1 byte. For example in Unicode characters are variable length. ✓

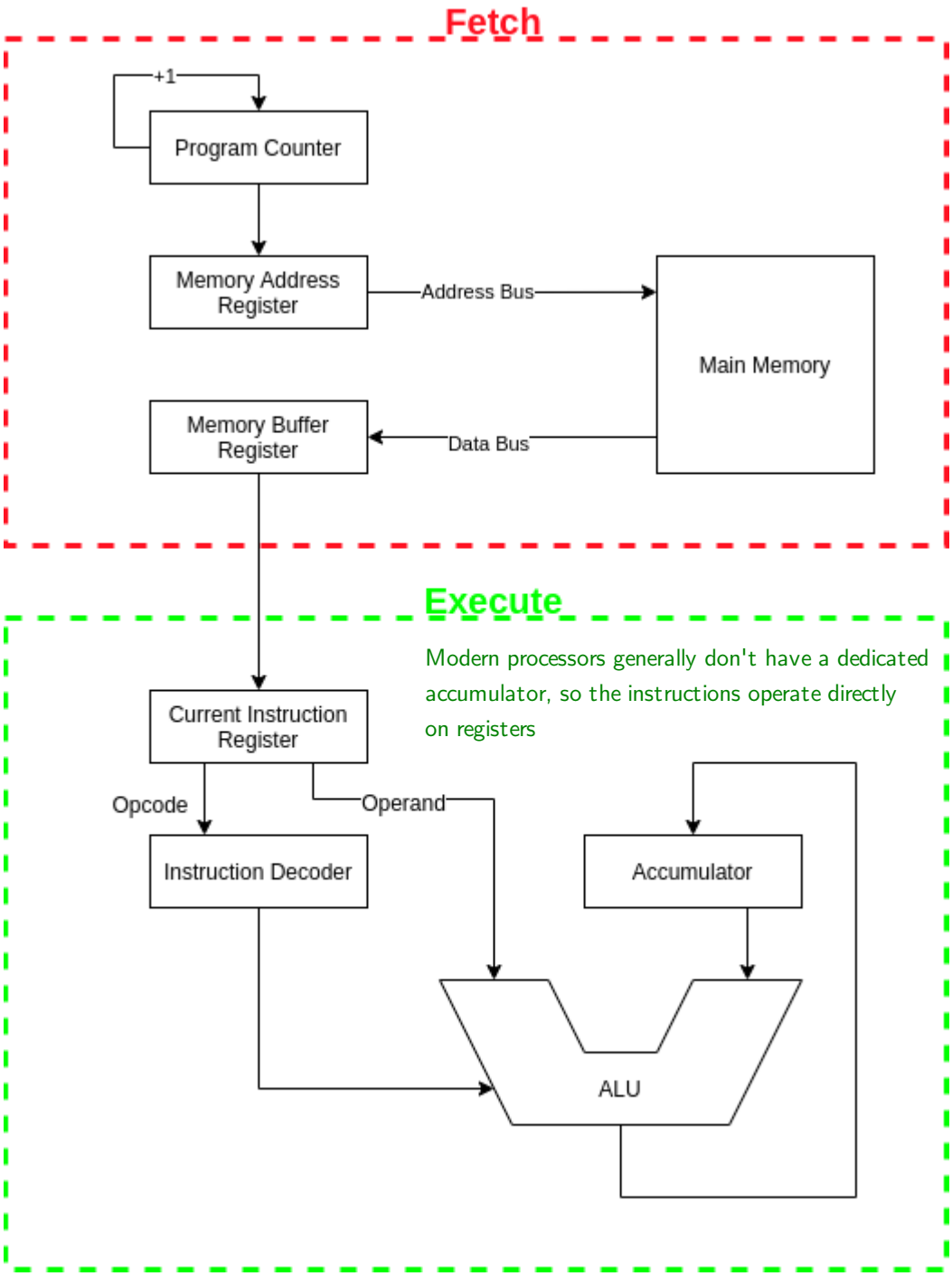
iv. An opcode followed by zero or more operands. The opcode specifies which process to carry out, and the operands specify the data on which to perform the process, or where to find that data. ✓

(b) If the sign-and-magnitude representation is being used for signed numbers, then yes. If the numbers 00000010 and 10000001 are to be added, and they are treated as unsigned numbers, the result should be 10000011 whereas if they are sign-and-magnitude signed numbers then the result should be 00000001. However, with two's complement, this is

not necessary for addition. Ignoring overflow, 00000001 + 11111111 should be 00000000 whethere they are considered signed or unsigned. ✓

(c) A context switch is when the operating system saves the state of one thread or process and loads the state of another. The processor can then start executing the newly loaded thread/process. ✓

3.



The PC contains the address in main memory of the next instruction to be executed. Its contents are copied to the MAR. The PC can then increment. The contents of the MAR is sent to the main memory via the address bus. The main memory fetches the instruction at that address and sends it back via the data bus. It is then stored in the MBR. Its contents are copied to the CIR so that the fetch cycle can continue while the instruction is being decoded and executed. The opcode is sent from the CIR to the instruction decoder which, depending

on which opcode is present, will send a control signal to the ALU (if this is indeed an arithmetic operation) which will act on both the operand of the instruction and the value stored in the accumulator. The result will then replace the value in the accumulator. ✓

4. Preemptive scheduling can be fairer, as if a process is taking up the CPU for a long period of time, it can be preempted so that other processes can run instead. However preemption is complicated to implement as you need a timer and you must worry about concurrency. Non-preemptive scheduling is simpler to implement, but it is prone to denial-of-service wherein one process hogs the CPU. Non-preemptive methods also often result in greater overall waiting times than preemptive methods. ✓

5.

   (a) The CPU will be allocated to the process in the ready queue with the highest priority. If multiple processes have the same priority, then many different techniques can be used to break the tie including first come first serve, shortest job first, shortest remaining time first, or round robin. ✓

   (b) FCFS is essentially a static priority scheduling algorithm where the priority is based on the time at which the process arrives (earlier processes have higher priority than later processes). ✓

   SJF is essentially a static priority scheduling algorithm where the priority is based on the estimated amount of time the job will take (shorter jobs have higher priority) ✓

   SRTF is essentially a dynamic priority scheduling algorithm where the priority is based on the estimated amount of time the job has remaining (shorter jobs have higher priority) however it is important to note that the priority queue here cannot simply be the ready queue, because the priority of the currently running process will update while it runs ✓ *

   Similarly RR is essentially a dynamic priority scheduling algorithm where the priorities of the jobs cycle giving each one a block of time during which it has the highest priority. However, again, the priority of the currently running process is also changing relative to those in the ready queue. ✓

   (c) Low priority processes are not guaranteed to run ever (starvation). This can be addressed by using dynamic priority scheduling. The priority of a process can increase over time until it is run. ✓

   (d) I/O intensive jobs are blocking. Requests to I/O devices can take a relatively long time, and during that time other processes can be run. As such it is useful to run the I/O intensive processes first, so that other processes can be run while they are blocked. ✓

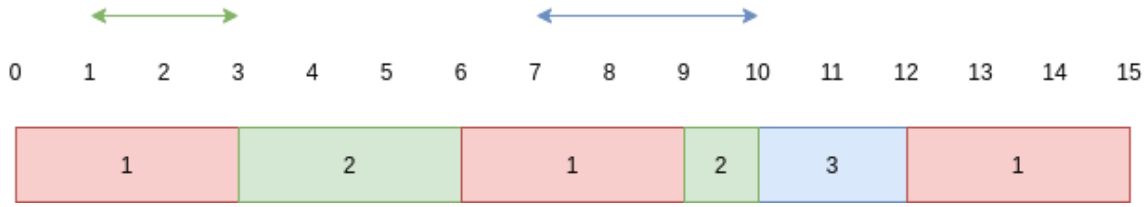
6.

   (a) It is preemptive ✓

   (b) One user cannot use up the CPU for a long period of time, preventing the other users from using it at all. ✓

* I'm not sure what you mean by the priority queue being different from the ready queue: as long as no processes arive, priorities don't change in any way that matters, and if you preempt what's currently running on each arrival, you only need to look at what's in the ready queue (including the preempted process).

(c)



P_1 waits 0

P_2 waits 2

P_3 waits 3

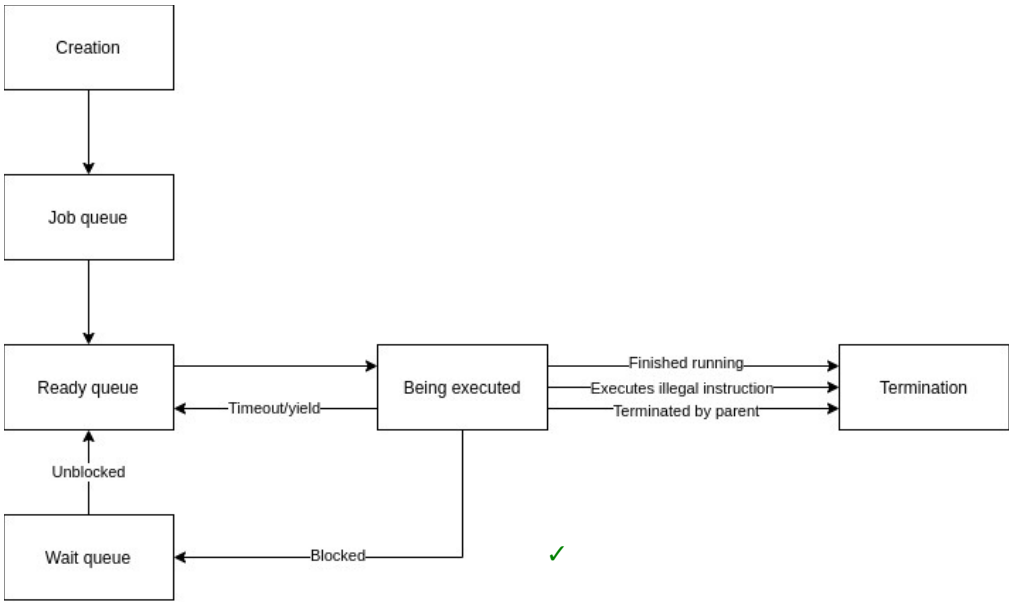
Average waiting time = 5/3 = 1.667 ✓

(d) Advantage: Faster response time

Disadvantage. More wasted time spent context switching ✓

7.

(a)



✓

When a process is created, it gets added to the job queue. From there it is admitted to the ready queue. Processes in the ready queue are in main memory and are ready to be executed. According to the scheduling algorithm, it is eventually allocated to the CPU. If a timer runs out (preemptive scheduler) or the process yields, it is put back in the ready queue. If it blocks (e.g. waiting for an I/O device) then it is put into the wait queue. Once it is unblocked (e.g. the I/O device response has arrived) then it gets put back into the ready queue.

The process terminates when either it has executed all of its instructions, it attempts to execute an illegal instruction, or it is killed by its parent. ✓

(b) Information about every process such as the process number, the current process state, the CPU scheduling information, the process context (the program counter and the contents of the CPU registers) and memory management information.

✓

(c) They require estimates of how long a process will take to complete (burst length). This can be obtained using exponential averaging over previous CPU burst lengths. ✓

(d) Advantage: simpler to implement as you don't need a timer

Disadvantage: prone to denial-of-service ✓

(e) When an I/O device has data or a DMA data transfer is completed, the CPI receives an interrupt. The process which was waiting for the data is then moved from the wait queue to the ready queue and the scheduler determines when the CPU should get allocated to it again. At this point the state of the registers when the request was sent is restored and the process can use the received data. ✓

(f) There is a risk that the data on the bus can be corrupted by multiple devices trying to output to it at once. If this happens, there's something wrong with your hardware :)

2012p2q3

a)

i. The contents of the program counter are copied to the process control block along with the contents of the CPU registers. When the context is restored, the program counter is restored from that stored value and the process can continue executing

ii. General purpose registers, processor status register

b)

i. j, i. The scheduler does not need to run

ii. j, g. The scheduler does need to run

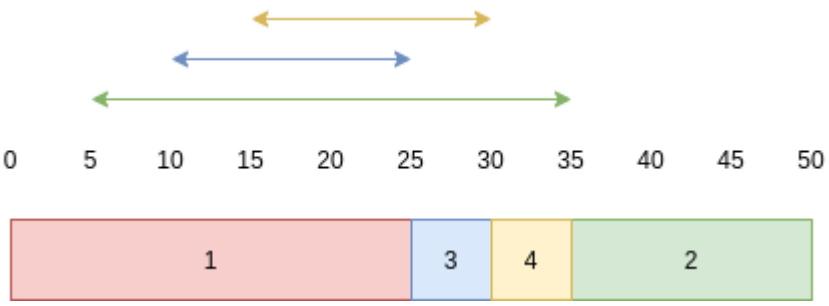
iii. b, a. The scheduler does not need to run

iv. b, c. The scheduler does not need to run until the I/O operation is complete at which point transition e occurs and the scheduler needs to be run

2010p2q3

b)

i.



P_1 waits 0.

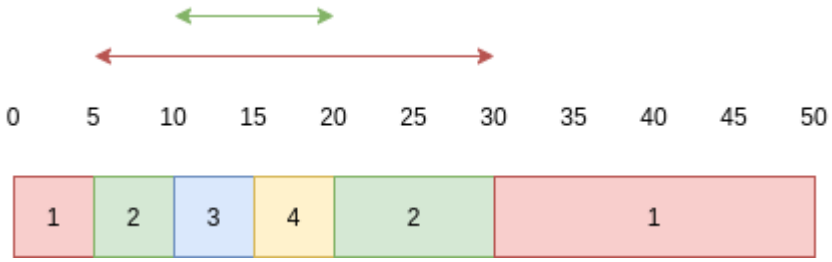
P_2 waits 30

P_3 waits 15

P_4 waits 15

Average waiting time = 15 ✓

ii.



P_1 waits 25

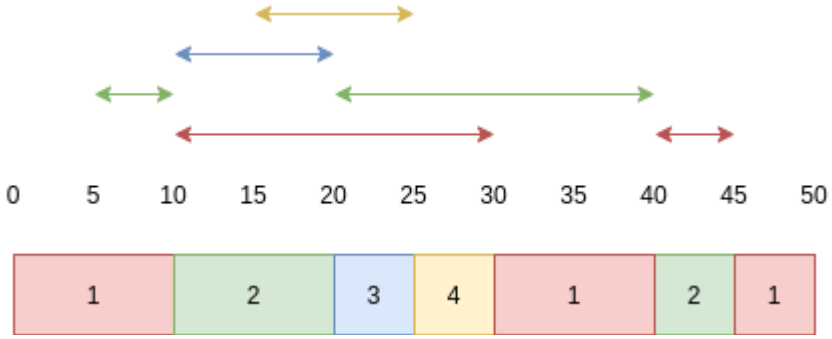
P_2 waits 10

P_3 waits 0

P_4 waits 0 ✓

Average waiting time = 8.75

iii.



P_1 waits 20 + 5 = 25

P_2 waits 5 + 20 = 25

P_3 waits 10

P_4 waits 10

Average waiting time = 17.5

^-- what happens here depends on what you do with new arrivals: if you add them to the start of the queue you get this behaviour, but the more common choice is to add them to the end, so 1 would get scheduled again before 4 gets to run.

iv. The advantage is that the average response time will go down. However the disadvantage is that this requires a greater proportion of the CPU's time to be wasted while the context switches. ✓