

CIS 450 – Computer Architecture and Operations  
Homework #5  
(25 points)

Due: Wed., Apr. 18, 2018, by 11:59 pm

~~fork()~~ fork()

parent == 1  
child == 0  
→ ~~child~~ → ~~parent~~

1. (5 pts.) Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

parent  
child

```
int main () {
    if (fork() == 0) {
        if (fork() == 0) {
            printf("3"); fflush(stdout); //child of a child
        }
        else {
            pid_t pid; int status;
            if ((pid = wait(&status)) > 0) {
                printf("4"); fflush(stdout);
            }
        }
    }
    else {
        printf("2"); fflush(stdout);
        exit(0);
    }
    printf("0"); fflush(stdout);
    return 0;
}
```

fork()

→ parent makes a child

For each of the following strings, circle whether (Y = yes) or not (N = no) this string is a possible output:

- a. 32040 Y N  
b. 34002 Y N  
c. 30402 Y N  
d. 23040 Y N  
e. 40302 Y N

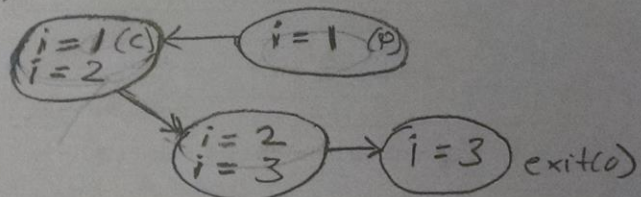
2. (10 pts.) Process creation questions:

- a. How many processes are created, including the parent process, when the following code is executed? 4.

```
int main()
{
    int i, p;
    for(i=1; i<=3; i++)
    {
        p = fork();
        if (p>0)
        {
            printf("i = %d\n", i);
            exit(0); // this one (*)
        }
        printf(("i = %d\n", i);
    }
    exit(0);
}
```

Process Model

b)



- b. Draw the Process Model above for the processes created in part (a), including the parent process, in the process model, *indicate the output generated by each process*.  
c. If the first exit(0) statement in part (a) is removed, denoted with (\*), how many processes would be created? 8. You don't need to draw the process model for this part ;-).



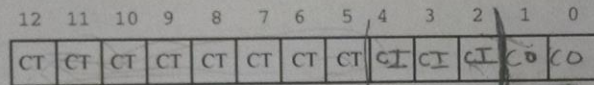
3. (10 pts.) The following problem concerns basic cache lookups. You may assume that:

- The memory is byte addressable. Memory accesses are to 1-byte words, not 4-byte words.
- Physical addresses are 13 bits wide.
- The cache is a 2-way set associative cache with a 4-byte line size and 16 total lines as shown below. Note that all numbers are given in hexadecimal format.

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	99	04	03	48
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	2F	81	FD	09	0B	0	8F	E2	05	BD
3	06	0	3D	94	9B	F7	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	B0	39	D3	F7
6	91	1	A0	B7	26	2D	F0	0	0C	71	40	10
7	46	0	B1	0A	32	0F	DE	1	12	C0	88	37

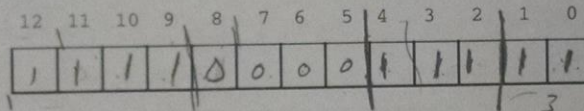
- (a) The box below shows the format of a physical address. Indicate, by labeling the rest of the fields in the diagram, the bits that are used to determine the following:

CT = Cache Tag (done), CI = Cache Index, and CO = Cache Offset



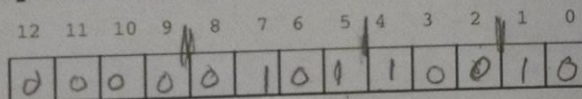
- (b) For each physical address given below, if a cache hit occurs, indicate the cache entry accessed and the cache byte value returned in hex. If a cache miss occurs, just write 'N' next to "Cache Hit?" and leave the cache byte returned blank.

Physical Address: 0x1E1F



Parameter	Value
Byte Offset	0x03
Cache Index	0x07
Cache Tag	0xF0
Cache Hit (Y/N)?	N
If Hit, Cache Byte Returned	0x

Physical Address: 0x00B2



Parameter	Value
Byte Offset	0x02
Cache Index	0x04
Cache Tag	0x05
Cache Hit (Y/N)?	Y
If Hit, Cache Byte Returned	0xC2