Solution

Problem 1: (7 points)

1. ab0c ba0c
2. acb1c abc1c bac1c

Problem 2: (10 points)

1. process#2
2. No.

   Because the execve function is called once and never returns, only except that it returns to the calling program if there is an error. But this situation is already handled and the process will exit. So character "C" will never be printed out.

3. Not sure.

   It is not deterministic.

   Because "A" is printed in process#3, "B" is printed in process#1, process#3 is originally a child of process#2, as process#2 exits immediately after it forks process#3. So the waitpid in process#1 will not wait process#3 to exit because process#3 is not a child of process#1. The sequence will not be deterministic.

4. INIT process.

Problem 3: (21 points)

1. load (%rax, %rdx.n-1, 4) -> t.n

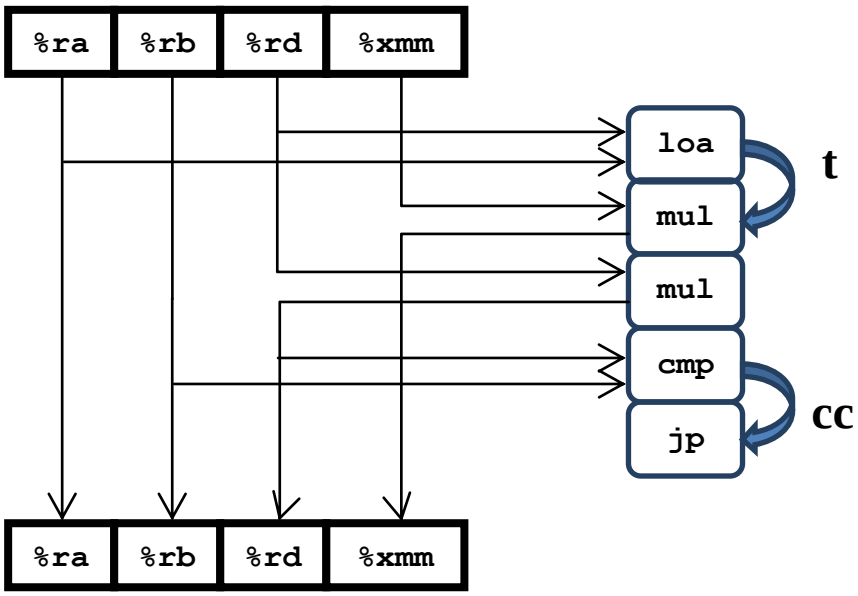
   mul t.n %xmm0.n-1 -> %xmm0.n

   mul $2, %rdx.n-1 -> %rdx.n

   cmpq %rdx.n, %rbp -> cc.n

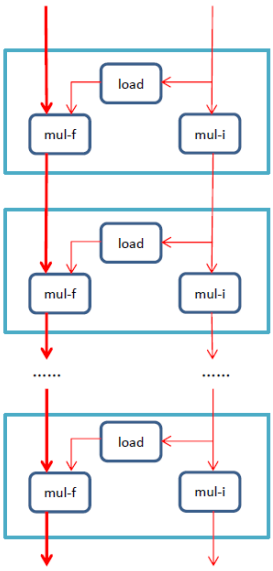
   jp .Loop

2.



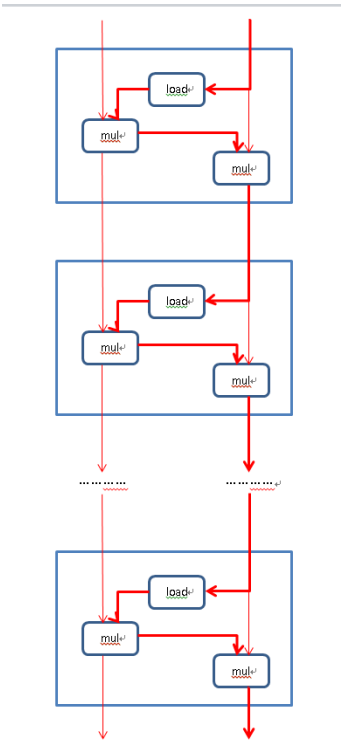
3. Read-only: %rax, %rbp (1′)
   Write-only: (1′)
   Local: t, cc (1′)
   Loop: %rdx, %xmm0 (1′)

4. 3′ for the graphic and 2′ for the critical path.



5. The CPE in Question.4 is 3. (2′)

6. If the instruct at line 3 is replaced, the critical path is changed



The CPE now should be 7, because of the two data dependency, between the two muls and between the load and mul. So 1 for load, 3 for each mul, the CPE is 7. (4′)

Problem 4: (8 points)

1. No, for n = 1,3,5,7,9….

2.
```
int fac_u2(int n, int *data) {
    . . .
    /*remaining element*/
    for(;i<n; i++)
        result *= data[i];
    return result;
}
```

3. Reassociation Transformation and parallel.

Problem 5: (6 points)

1. 2*8 + 8*8*8 inner most loop a[][] will get 2 cache misses while b[][] will be missed everytime

2. by 4x4 blocking a[][] still missed 2 times, while b[][] each 4*4 chunk miss cache 4 times. 4 chunks misses 4*4

Thus result is 2*8 + 4*4*8

Problem 6: (48points)

1. (12')

Stage	Generic	Specific
	rjne rB	rjne %ebx
Fetch	icode:ifun ← M_1[PC] = E:4 rA:rB ← M_1[PC+1] = F:rB valP ← PC + 2	icode:ifun ← M_1[PC] = E:4 rA:rB ← M_1[0x01c] = F:3 valP ← 0x01d
Decode	valB ← R[rB]	valB ← R[%ebx] = 0x023
Execute	Cnd ← Cond(CC, ifun)	Cnd ← 0
Memory	--	--
Write Back	--	--
PC Update	PC ← Cnd?valB:valP	PC ← 0?0x023:0x01d = 0x01d

2. (10')

Condition	Trigger
Target-addr Hazard	IRJXX in {D_icode}
Mispredicted Branch	E_icode = IRJXX && !e_Cnd

Condition	F	D	E	M	W
Target-addr Hazard	S	B	N	N	N
Mispredicted Branch	N	B	B	N	N

3. (12')

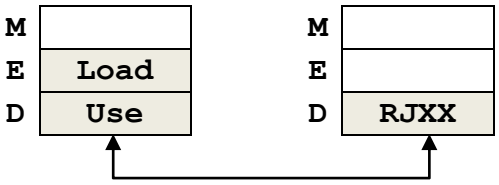
```
int f_pc = [
    # Mispredicted branch.  Fetch at incremented PC
    M_icode == IRJXX && !M_Cnd : M_valA;
    # Stalling at fetch while rjXX passes through pipeline
    E_icode == IRJXX : E_valB
];

bool D_bubble =
    # Mispredicted branch
    E_icode == IRJXX && !e_Cnd)||
```

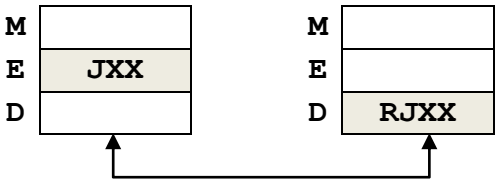
```
        # Stalling at fetch while rjXX passes through pipeline
        IJRXX in {D_icode};

    bool E_bubble =
        # Mispredicted branch
        (E_icode == IRJXX && !e_Cnd); // or nothing!
```

4. (9')



Condition	F	D	E	M	W
Load	Stall	Stall	Bubble	Normal	Normal
RJXX	Stall	Bubble	Normal	Normal	Normal
Combination	Stall	B+S	Bubble	Normal	Normal
Desired	Stall	Stall	Bubble	Normal	Normal



Condition	F	D	E	M	W
JXX	Normal	Bubble	Bubble	Normal	Normal
RJXX	Stall	Bubble	Normal	Normal	Normal
Combination	Stall	Bubble	Bubble	Normal	Normal

5. (3' + 2')

0.15 * 0.20 * 1 = 0.03
0.01 * 1.00 * 3 = 0.03
0.15 * 0.40 * 2 = 0.12
0.05 * 0.40 * 2 = 0.04
0.05 * 0.60 * 1 = 0.03
CPE = 1 + 0.03 + 0.03 + 0.12 + 0.04 + 0.03 = 1.25

0.15 * 0.20 * 1 = 0.03
0.01 * 1.00 * 3 = 0.03
0.15 * 0.60 * 2 = 0.18
0.05 * 0.60 * 2 = 0.06
CPE = 1 + 0.03 + 0.03 + 0.18 + 0.06 = 1.30