104 台大計架

1.
(a) stack machine 利用stack做 store).
(b) 太多的register造成loading時間很久(每一次切process要重新load)，所以適當即可
(c) 若切成 caller reg / callee reg 代表切換時，可同時切成callee reg => ∴ caller不用額外記錄調用callee的reg狀況，適合用在有子程序呼叫之program
(D) 使常用指令一般化，可使reg使用效率增加(因為有多種定址模式)
若只有一種set reg，則reg要記錄每種type(使效能較低且繁雜)

2.
(A) $2^{20}+2^{21}+2^{22}+2^{24}+2^{25}+2^{26}+2^{27}+2^{28}+2^{29}$ ※

(B) 0 01111110 1110……0 1.111×2^{-1} => 0.1111 => $\frac{15}{16}$ ※

(C) 00111 => luì ※

(D) Char 1byte => 8bit/一個 , 63 , 112 , 0 , 0 => "?p NiL NiL" => "?p" ※

(E) 在x86上看到的相同，AB是"?p"

(F) 捨入計算誤差. $a=2\times10^{38}$ $b=-2\times10^{38}$ $c=1$
若(a+b)+c => result=1
若a+(b+c) => result=0

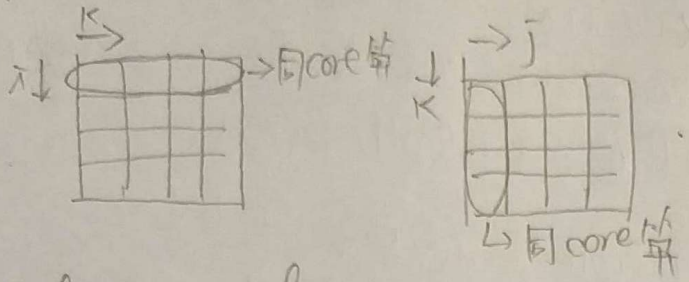
3.
(A) alway taken 跟 1 bit taken prediction
(B) 利用jr實現switch，或利用jr實現call return => jr $ra
(C) 因為 indirect branch 沒有確切位置，要到 run-time 時才能確定跳躍的位址，且非常耗能(stall)

4.
A = avg penalty ① (8+24)×0.08=2.56 ② (16+24)×0.04=1.6 ✓ ③ (32+24)×0.03=1.68
block size 16 的 avg penalty 最小

B= critical word first = 先fetch需要的word，剩下慢慢填入cache
則選miss rate最小的(需要的第一時間皆可拿到)，提高hit

5.
(A) Strong-scaling：可利用加processor增加計算效能
(B) 讓每個core算同一個block，不要平行到同block內資料

eg



6. NVM：有lower access latency
① 因為一般做I/O要花大量時間在等待低速的硬體讀寫，所以多用I/O driven的方式
② 若增加NVM方式降低I/O latency時間則可考慮用polling加快I/O request速度

7.
A. 若time quantum太小會造成cpu使用率低(因為switch的latency影響加大)如此一來多了很多不必要的額外時間，整體的turnaround time變長
B. 使process盡可能在同一個processor上執行可減少flush & load的額外時間也能使各cache不用重新load => MRS rate下降

8.

```
semphoe chair_num=4;
semphoe mutex=1;
int time[5]=0;
semphoe get_first=1
```

```
competing(process *p, int i)
  time[i]++;
  if(time[i]==3)
    wait(get_first);
      wait(chair_num);
        wait(chair_num);
          wait(mutex)
            C.S & time[i]=0;
          signal(mutex)
        signal(chair_num)
      signal(get_first);

  else
      if(someone time[i]==3)
        signal(get_first);
    wait(chair_num);
      wait(mutex)
        C.S & time[i]=0;
      signal(mutex)
    signal(chair_num)
```

9.

(A) 要使 sys 進入 thrashing 狀態要將可用 memory 全用滿再進行 process 的讀寫 ⇒ 造成 memory 不夠需不斷 page fault 自依然不足之狀態

(B) 若不能搶其他 process 所釋放的 frame 來用，在原 frame 上有 thrashing 現象之 process 仍會 thrashing ⇒ 在該 process memory 無法被滿足

(C) 利用 memory 使用局部、集中的特性達到偵測功能 ⇒ 計算所需要的 frame 再配置

(D) 沒辦法完全避免，假設有 caller function 會利用到額外堆疊呼叫，則 working set 不一定能偵測得到，仍有可能發生 thrashing

(E) 若一個 process 在 VM ⓐ 產生 thrashing，對他 VM 不會有任何影響，因 VM 確保各個 guest OS 上之 simulation Hardware 不會互相影響，給每個 VM 有固定 Memory.

10.

① 一般用 read/write I/O 是 linear I/O

② vector I/O 可在單一個 instruction 中做多個 buffer 空間做同時的 input/output 也可 SIMD 或收集多個 Data 集成單一數據流

③ 若 data flow 具有可平行化的切割方式 ⇒ 能夠做數據處理

④ 單一 vector I/O 可替換成多個 linear I/O.

⑤ 可避免執行時單元性問題