CS 551 Advanced Computer Architecture

Chihsiang Wang 101-64106
Assignment#1

A 11.
For 32-bits:
1. The size of foo struct is 1+1+4+8+2+4+8+4+4+4 = 40 bytes.

2. Consider the data alignment, 4 bytes for a chunk in 32-bits system:
char->padding[1], bool->padding[1], short->padding[2]
40+1+1+2 = 44 bytes is the minimun requirement for this struct.

3. sort the elements large to small in the struct. So doesn't need padding

```
struct foo{
double d;
double g;
char* cptr;
float* fptr;
int c;
int x;
float f;
short e;
char a;
bool b;
}
```

For 64-bits:
Similar to 32-bits, but data size is somehow different(long and pointer are double)
1. The size is 1+1+4+8+2+4+8+8+8+4 = 48 bytes.

2. 8 bytes for a chunk:
48+paddin[2]+padding[2]+padding[4] = 56 bytes.

3. Same with 32-bits.

A 18.
```
************************************************************
* I found some reference on the internet for this question *
* but I still try to do this by myself. *
************************************************************
```
```
A = B + C;
B = A + C;
D = A - B;
```

Accumulator architecture:
Means only one register

LOAD B ; Put B into Accumulator
ADD C ; Acc(B) + C, Acc = B + C
STORE A ; Access memory->Store Acc into Memory[A]
; A = B + C
ADD C ; Acc = B + C = A, then Acc = A + C
STORE B ; Acc = B
NEGATE ; Acc = (-)B
ADD A ; Acc = -B + A
STORE D ; D = A - B

Memory Access: 4 times
Register Access: 8 times

Memory-memory architecture:
ADD A, B, C ; Access memory directly Memory[A] = Mem[B] + Mem[C]
ADD B, A, C ; Memory[B] = Mem[A] + Mem[C]
SUB D, A, B ; Memory[D] = Mem[A] + Mem[B]

Memory Access: 9 times
Register Access: 0 times

Stack architecture:
I found this one from internet as reference

Push B ; TOS <- Mem[B], NTTOS <- *
Push C ; TOS <- Mem[C], NTTOS <- TOS
Add ; TOS <- TOS + NTTOS, NTTOS <- *
Pop A ; Mem[A] <- TOS, TOS <- *
Push A ; TOS <- Mem[A], NTTOS <- *
Push C ; TOS <- Mem[C], NTTOS <- TOS
Add ; TOS <- TOS + NTTOS, NTTOS <- *
Pop B ; Mem[B] <- TOS, TOS <- *
Push B ; TOS <- Mem[B], NTTOS <- *
Push A ; TOS <- Mem[A], NTTOS <- TOS
Sub ; TOS <- TOS − NTTOS, NTTOS <- *
Pop D ; Mem[D] <- TOS, TOS <- *

Memory Access: 9 times
Register Access: 16 times

Load-store architecture:
16 Register is pretty enough
LOAD R1, B ; Put Memory[B] to R1
LOAD R2, C ; Put Memory[C] to R2
; Now I can use Register to do calculous
ADD R3, R1, R2 ; R3 = R1 + R2 = B + C
Add R1, R3, R2 ; R1 = R3 + R2 = A + C
Sub R4, R3, R1 ; R4 = R3 - R1 = A - B
Store A, R3 ; Memory[A] = R3
Store B, R1 ; Memory[B] = R1

Store D, R4 ; Memory[D] = R4
CLR R1 ; initialize R1
CLR R2 ; initialize R2
CLR R3 ; initialize R3

Memory Access: 5 times
Register Access: 15 times(include 3 times clear register)

c.
In this question, Accumulator architecture may be the most efficiency choice.
Then Load-store architecture, and Stack architecture is worst.

d.
Accumulator architecture:
Short instructions, but only good for small question,
when question getting bigger, the memory traffic will be really busy.

Memory-Memory architecture:
Always need to access memory, maybe it's low-expense design?

Satck architecture:
Short instructions, and Simple to use, but the stack register can not be choose
need to use carefully.

Load-Store architecture:
instructions can be fixed length and short, the design allows pipelining, makes it faster.