Exame recurso de Arquitetura de Computadores *versão A*

7/7/2018 Duração 3h *sem consulta*

As perguntas de escolha múltipla têm de ser respondidas na folha de respostas anexa.
Respostas erradas descontam 1/5 da cotação da pergunta ou alínea.

Número: _____ *Nome:* _____

1- (0,8 val) Considere uma representação de inteiros **com** sinal, complemento para 2, com 7 bits.

a) Qual o menor valor que é possível representar nesta configuração?

A. 0 B. –63 C. –64 D. –127 E. –128

b) Qual é o maior valor que é possível representar nesta configuração?

A. 63 B. 64 C. 127 D. 128 E. 256

2- (0,8 val) Considere uma representação de inteiros **sem** sinal com 7 bits.

a) Qual o menor valor que é possível representar nesta configuração?

A. 0 B. –63 C. –64 D. –127 E. –128

b) Qual é o maior valor que é possível representar nesta configuração?

A. 63 B. 64 C. 127 D. 128 E. 256

3- (0,5 val) Considere uma representação do tipo `float` em 32bits como no IEEE 754 precisão simples. Relativamente aos valores reais que pode representar corretamente no seu programa (C ou Java) genericamente sabe que têm as seguintes limitações (assinale a correta):

A. Só se pode representar reais positivos.

B. Pode-se representar todos os reais num intervalo que é definido, aproximadamente, pelo número de bits usados para expoente (aprox. de -10^{38} a 10^{38})

C. Pode-se representar todos os reais, entre as representações de *-infinito* e *+infinito*.

D. Pode-se representar todos os reais que tenham menos de, aproximadamente, 11 casas decimais.

E. Só se pode representar alguns reais num intervalo que é definido, aproximadamente, pelo número de bits usados para expoente (aprox. de -10^{38} a 10^{38})

4- (0,5 val) Considere uma representação de inteiros com sinal de 7 bits, de um CPU onde foi realizada a operação aritmética "60 + 20". Qual dos seguintes termos corresponde ao valor das flags (Carry, Overflow, Sign, Zero)?

A. CF=0, OF=1, SF=1, ZF=0

B. CF=0, OF=0, SF=0, ZF=0

C. CF=1, OF=1, SF=1, ZF=0

D. CF=1, OF=0, SF=0, ZF=0

E. CF=0, OF=1, SF=0, ZF=1

5- (0,5 val) Considere a seguinte sequência de instruções *assembly* Intel/Linux

```
        movl $3,  %eax
        movl $1,  %ebx
        sub  %ebx, %eax
        ja   L1
        jmp  L2
L1:     movl %ebx, %eax
L2:     dec  %ebx
```

Escreva na caixa ao lado o conteúdo dos registos e das *flags*, após a execução de toda a sequência de instruções.

eax = 1

ebx = 0

ZeroF = 1
OverflowF = 0
SigneF = 0
CarryF = 0

6- (1 val) Considere uma representação de números em vírgula flutuante segundo a norma IEEE754 para precisão simples:

- 1 bit = sinal
- 8 bits = expoente somado de 127
- 23 bits = parte fracionária da mantissa (o valor efetivo da mantissa é 1.XXX...XXX)

a) Imagine que num registo tem um valor do tipo float que, em binário, corresponde a: 01111111 00000000 00000000 00000000_2 (= 0x7f000000). Ao multiplicar por 2 obteve o seguinte resultado: 01111111 10000000 00000000 00000000_2 (= 0x7f800000). Qual das seguintes afirmações é verdadeira para esta operação realizada no âmbito da norma IEEE754 para precisão simples?

A. A operação dá origem a um resultado que está correto.

B. A operação dá origem ao valor *+infinito* devido a um overflow positivo.

C. A operação dá origem ao valor *-infinito* devido a um overflow negativo.

D. A operação dá origem a um resultado que está correto apesar de ter existido um overflow positivo.

E. A operação dá origem a um resultado incorreto devido a um underflow positivo.

b) Indique agora que valor real corresponde à seguinte representação binária nesta norma:
11000011 00100000 00000000 00000000_2

A. -160.0 B. -160.25 C. -192.0 D. -192.25 E. -192.5

7- (0,5 val) Considere o seguinte excerto de código C:

```
int x = 4;
int y = x + 6.7;
```

Que resultado fica guardado na variável y?

A. 11

B. 10

C. 10.7

D. Nada, pois a compilação do programa produz o erro "int e double são tipos incompatíveis"

E. 0

8- (0,5 val) Qual é a forma correta de ler valores para as variáveis "a" e "b" no seguinte trecho de um programa em C?

```
#include<stdio.h>
float a;
int b;
```

A. `scanf("%f %f", &a, &b);`

B. `scanf("%f %d", &a, &b);`

C. `scanf("%d %d", &a, &b);`

D. `scanf("%d %f", a, b);`

E. `scanf("%f %d", a, b);`

9- (0,5 val) Qual das seguintes expressões em C permite extrair o valor do bit de sinal de um inteiro X (tipo `int`) de 32 bits, ou seja, obter 0 ou 1, consoante o estado desse bit?

A. X >> 32

B. X >> 31

C. (X >> 31) & 1

D. (X & 1) << 32

E. (X & 1) << 31

10- (0,5 val) Qual o output do seguinte trecho de um programa em C?

```
#include<stdio.h>
void main () {
  printf ("%d", -1<<4);
}
```

A. fffffff0 B. -16 C. -31 D. 16 E. 0fffffff

11- (0,5 val) Considere o seguinte o programa em Assembly para Intel, 32bis, little-endian.

```
.data
X      .int 0
.text
       movl    $512, X
```

Assumindo que a variável X está guardada nos endereços 2000 a 2003, qual dos seguintes termos representa o conteúdo das 4 posições de memória de 2000 a 2003 (por esta ordem) depois da execução da instrução indicada?

A. (0, 0, 1, 0)
B. (0, 2, 0, 0)
C. (1, 1, 0, 0)
D. (0, 0, 2, 0)
E. (0, 0, 1, 1)

12- (0,5 val) No contexto da arquitectura Intel IA-32 a instrução **pop %ebx**

A. Retira o conteúdo de %ebx da pilha de execução do programa, mantendo %esp com o mesmo valor
B. Coloca o conteúdo do registo %ebx no topo da pilha de execução do programa
C. Retira o conteúdo do topo da pilha de execução do programa e coloca-o no registo %ebx
D. Retira o conteúdo do topo da pilha de execução do programa e coloca-o na posição de memória cujo endereço está guardado no registo %ebx
E. Coloca no topo da pilha de execução o conteúdo da posição de memória cujo endereço está guardado no registo %ebx

13- (0,5 val) No contexto da arquitectura Intel IA-32, qual das seguintes frases descreve de forma mais completa as ações que são efetuadas ao executar a instrução **call fun1** ?

A. Coloca no %eip o endereço fun1 permitindo executar uma subrotina.
B. Coloca no %esp o endereço fun1 permitindo executar uma subrotina.
C. Coloca o valor em %eip no topo da pilha e o endereço fun1 em %eip, permitindo executar uma subrotina.
D. Coloca o valor em %esp no topo da pilha e o endereço fun1 em %esp, permitindo executar uma subrotina.
E. O CPU executa a primeira instrução da função fun1 da linguagem C.

14- (0,5 val) Considere os seguintes excertos de código assembly IA-32, onde **jg** (Jump if Greater) opera sobre números com sinal e **ja** (Jump if Above) opera sobre números sem signal:

```
mov $255, %al          mov $255, %al
cmp $0, %al            cmp $0, %al
ja Etiq                jg Etiq
```

A. O salto para Etiq é apenas efectuado pela execução da instrução jg
B. O salto para Etiq é efectuado em ambos os códigos
C. O salto para Etiq não é efectuado em qualquer dos códigos
D. O salto para Etiq é apenas efectuado pela execução da instrução ja
E. Há um erro pois as instruções para saltos com registos de 8 bits não são ja e jg, mas sim ja.b e jg.b

15- (0,5 val) Nos processadores multi-core:

A. O motor de execução (registos, ALU, pipeline de execução, etc) é partilhado por todos os cores.
B. Cada core tem a sua própria memória central.
C. Cada core tem o seu próprio conjunto de registos mas a ALU e o pipeline de execução são partilhados.
D. Cada core tem o seu próprio conjunto de registos e ALU, mas o pipeline de execução é partilhado.
E. Cada core tem o seu próprio motor de execução (registos, ALU, pipeline de execução, etc).

16- (0,8 val) As arquiteturas de computador usam memórias cache (um ou mais níveis) entre o CPU e a memória central.

a) Qual é o objectivo destas caches?

A. Aumentar a memória disponível para os programas
B. Reduzir o tempo de acesso à memória
C. Permitir endereços virtuais e paginação
D. Permitir que o CPU tenha registos com mais bits de dimensão
E. Tornar o bus de acesso à memória central mais rápido

b) O *hitratio* nessas caches e, em consequência, a sua eficácia, aumenta quando os acessos à memória central têm que propriedades?

A. Acede-se aos mesmos endereços num curto intervalo de tempo (localidade temporal); mas não se acedem a endereços próximos num curto intervalo de tempo (localidade espacial)
B. Não se acedem aos mesmos endereços num curto intervalo de tempo (localidade temporal); mas acede-se a endereços próximos num curto intervalo de tempo (localidade espacial)
C. Acede-se aos mesmos endereços num curto intervalo de tempo (localidade temporal); e acede-se a endereços próximos num curto intervalo de tempo (localidade espacial)
D. Não se acedem aos mesmos endereços num curto intervalo de tempo (localidade temporal); mas acede-se a endereços aleatórios da memória RAM (Random Access Memory)
E. Não se acedem a endereços próximos num curto intervalo de tempo (localidade espacial); mas acede-se a endereços aleatórios da memória RAM (Random Access Memory)

17- (1 val) Uma determinada arquitetura dispõe de uma cache de 256 Kbytes, associativa por grupos. Estando esta organizada em grupos (sets) de 4 linhas onde cada linha contém 64 bytes.

a) Quantos grupos (sets) existem nesta cache?

A. 512 B. 1024 C. 2048 D. 4098 E. nenhuma das anteriores

b) De que forma um endereço é procurado na cache?

A. É identificado o grupo e chave do endereço; verifica se o grupo aparece dentro da entrada dada pela chave.
B. É identificado o grupo, chave e deslocamento na linha; usa o deslocamento na cache para verificar se a linha tem a chave igual.
C. É identificado o grupo e chave do endereço; verifica se no grupo respectivo da cache aparece essa chave e o bit de validade é 1.
D. É identificado o deslocamento e a chave do endereço; verifica se essa chave aparece na cache e se o seu bit de validade é 0.
E. É identificado o grupo do endereço; verifica se nesse grupo da cache tem o bit de validade a 1.

c) Indique como o endereço seguinte (representado em binário) é interpretado, em termos de deslocamento, grupo e chave, para ser procurado na cache?

```
00000000 00101000 00000001 10000010
```

A. desl=2; grp=6; chv=40
B. desl=0; grp=40; chv=2
C. desl=2; grp=12; chv=70
D. desl=0; grp=10; chv=3086
E. desl=2; grp=3; chv=10

18- (0,5 val) Como se chama o mecanismo que nas arquiteturas do hardware permite a execução de vários programas independentes, cada um com um espaço de endereços privado e, em alguns casos, até maior que a memória RAM instalada?

A. Memória cache
B. Page-fault
C. Memória virtual
D. TLB (Translation Lookaside Buffer)
E. Memória RAM (Random Access Memory)

19- (1,1 val) Uma arquitectura com endereços virtuais de 32 bits e com páginas com a dimensão de 1MBytes (2^{20}).

a) Qual o número máximo de entradas que pode ter a tabela de páginas de um processo?

A. 4 M entradas (2^{22}) B. 4 K entradas (2^{12}) C. 1 M entradas (2^{20})
D. 1 K entradas (2^{10}) E. nenhuma das anteriores

b) Para um dado processo em execução, o conteúdo da TLB na MMU é o seguinte (endereços em hexadecimal):

TLB:

página virtual	página física
8	0x00
1	0xAF
2	0xAD
6	0xEA

Indique se, conhecendo unicamente o conteúdo do TLB acima indicado, é possível obter o endereço real para cada um dos acessos aos endereços virtuais que se seguem (circule a palavra Não ou Sim). Em caso afirmativo, indique qual o endereço real obtido:

`0x00200001` → Não / Sim -> endereço real = `0xAD00001`

`0x00610000` → Não / Sim -> endereço real = `0xEA10000`

`0x00580111` → Não / Sim -> endereço real =

`0x0000029A` → Não / Sim -> endereço real =

c) Nos casos em que não é possível obter logo na TLB o endereço real, indique que ações o *hardware* toma para tentar obter esse endereço e em que condições produz um *page-fault*.

A. Verificando se o endereço existe na cache e, se não, consulta a tabela de páginas em memória. Caso não encontre produz um page-fault.
B. Verifica se a página está na tabela de páginas em memória e, se não, lê do disco (zona de swap) a página em falta. Caso não encontre produz um page-fault.
C. Verifica se a página está na tabela de páginas em memória e, se não, produz um page-fault.
D. Verifica se a entrada da página na tabela de páginas em memória é válida (bit validade=1), se não, produz um page-fault.
E. Verifica se a entrada da página na tabela de páginas em memória foi escrita (dirty bit=1), se não, produz um page-fault.

20- (0,5 val) Qual é o output produzido pelo seguinte programa em C?

```
#include<stdio.h>
int main() {
    char *p;
    p="%d\n";
    p++;
    p++;
    printf(p-2, 23);
    return 0;
}
```

A. 21 B. 23 C. 25 D. Segmentation fault E. *Nenhum output*

21- (0,5 val) Para executar as instruções "reservadas" (como por exemplo "in", "out", ...) o processador necessita de estar em modo:

A. Supervisor B. Administrador C. Utilizador D. Reservado E. Seguro.

22- (0,5 val) Qual é o output produzido pelo seguinte programa em C, sabendo que foi introduzido o valor 25 no teclado para o `scanf`?

```
#include<stdio.h>
int main() {
    int i;
    printf("%d\n", scanf("%d", &i));
    return 0;
}
```

A. 25 B. 2 C. 1 D. *Segmentation fault* E. *Nenhum output*

23- (0,5 val) Qual é o efeito do seguinte trecho de um programa em C?

```
FILE *fp;
fp = fopen("text.txt", "r");
```

A. Abrir o ficheiro "text.txt" já existente para leitura ou devolver NULL se o ficheiro não existir.
B. Abrir o ficheiro "text.txt" para leitura, criando-o vazio se não existir.
C. Abrir o ficheiro "text.txt" já existente para leitura e escrita ou devolver NULL se o ficheiro não existir.
D. Abrir o ficheiro "text.txt" já existente para leitura e escrita. Caso não exista, criá-lo vazio e abri-lo para leitura e escrita.
E. Caso o ficheiro "text.txt" não exista, criá-lo vazio e abri-lo para leitura e escrita. Caso já exista, reinicializá-lo a vazio e abri-lo para leitura e escrita.

24- (0,5 val) Qual das seguintes afirmações referentes à linguagem de programação C é verdadeira?

A. A invocação da função `fprintf(f, "%s","abc")` é válida e imprime "ABC" no ficheiro representado por f.
B. A invocação da função `printf("%S","abc")` é válida e imprime "ABC" no último ficheiro aberto com fopen.
C. A invocação da função `printf("%d",10)` é válida e imprime "10" no último ficheiro aberto com fopen.
D. Um ficheiro de texto é uma sequência de caracteres, onde cada linha é composta por zero ou mais caracteres seguidos de um caráter de mudança de linha ('\n').
E. Um ficheiro inicialmente criado como ficheiro de texto, posteriormente só pode ser aberto em modo texto e nunca em modo binário.

25- (0,5 val) Indique qual das seguintes frases descreve melhor como o processador da arquitetura Intel IA-32 se comporta a quando de uma interrupção, do hardware ou de software (instrução INT):

A. O processador vai executar as instruções da rotina de tratamento da interrupção.
B. O processador guarda o estado corrente e vai executar as instruções da rotina de tratamento da interrupção.
C. O processador muda de modo de execução e vai executar as instruções da rotina de tratamento da interrupção.
D. O processador muda de modo de execução, guarda o estado corrente e vai executar as instruções da rotina de tratamento da interrupção.
E. O processador muda de modo de execução, guarda o estado corrente, guarda a cache e tabela de páginas e vai executar as instruções da rotina de tratamento da interrupção.

26- (0,5 val) Uma operação de transferência de dados com um dispositivo por DMA (Direct Memory Access) oferece que vantagem?

A. Permite desencadear a transferência de um bloco de bytes sem o que dispositivo tenha de receber/enviar esses dados.
B. Permite a transferência de um bloco de bytes com o dispositivo sem que o CPU (programa) tenha de receber/enviar cada um desses bytes
C. Permite desencadear a transferência de um bloco de bytes, onde a cada interrupção o dispositivo recebe/envia cada um desses bytes.
D. Permite a transferência de um byte, sendo que o dispositivo acede diretamente à memória para receber/enviar esse byte.
E. Permite a transferência de um byte, sendo que temos interrupções para o dispositivo aceder diretamente à memória para receber/enviar esse byte.

27- (0,5 val) Numa arquitetura com controlador de interrupções, uma rotina de tratamento de uma interrupção, quando em execução … (complete com a frase correta)

A. é sempre interrompida quando ocorre uma nova interrupção de hardware.
B. nunca pode ser interrompida quando ocorre uma nova interrupção de hardware.
C. nunca pode ser interrompida quando está a aceder à memória.
D. é sempre interrompida quando ocorre uma nova interrupção, desde que o sistema de interrupções não esteja inibido pela instrução CLI.
E. pode ser interrompida quando ocorre uma nova interrupção mais prioritária, desde que o sistema de interrupções não esteja inibido pela instrução CLI.

28- (2 val)

a) Pretende-se que implemente uma função na linguagem C que, dada uma cadeia de caracteres, devolva o carácter com o código ASCII mais elevado. Os exemplos seguintes representam o resultado da invocação da função para diferentes cadeias:

```
printf("maior:%c\n", maiorCar("marte"));
printf("maior:%c\n", maiorCar("Abril"));
printf("maior:%c\n", maiorCar("EXIT"));
```

Output

```
maior: t
maior: r
maior: X
```

```
char maiorCar(char *cad) {

        char maior = '\0';   //considerando apenas os códigos ASCII (7bits)
        while ( *cad != '\0' ) {
                if ( *cad>maior )  maior = *cad;
                cad++;
        }
        return maior;
```

b) Implemente agora em *assembly* a chamada da função C anterior passando como argumento a cadeia `str` e guardando o resultado em `res`.

```
.data
      str  .ascii  "marte\0"
      res .byte  ' '
.text
# chamar maiorCar("marte") guardando o resultado em res

      push $str
      call maiorCar
      add $8, %esp
      movb %al, (res)
```

c) Implemente agora em *assembly* uma função idêntica à **maiorCar** que possa ser chamada a partir de um programa em C, tal como nos exemplos na alínea a).

```
        .global maiorCar
maiorCar:
        push %ebp
        mov %esp, %ebp
        mov 8(%ebp), %edx
        mov $0, %eax
ciclo:
        movb (%edx), %ecl
        cmp $0, %ecl
        je fim
        cmp %eal, %ecl
        jbe cont
        movb %ecl, %eal
cont: inc %edx
        jmp ciclo
fim:
        pop %ebp
        ret
```

29- (1 val) Implemente em C a função "`void rotacao(char *str, int salto)`". Esta função recebe uma string e realiza uma deslocação "à esquerda" dos seus caracteres num determinado número de posições indicado em *salto*, colocando no fim os caracteres que saem pelo inicio. Note que *salto* é sempre um valor positivo e menor que o comprimento da string. Como exemplo, veja o seguinte programa e a sua saída, onde se desloca os caracteres em 3 posições:

```
#include<stdio.h>
#include<string.h>
int main() {
  char *str = "abcdefghi";
  printf ("ORIG: %s\n", str);
  rotacao (str, 3);  // "roda" a string no buffer em 3 posições
  printf ("ROT[%d]: %s\n", 3, str);
}
```

Saída:
```
ORIG: abcdefghi
ROT[3]: defghiabc
```

```
void rotacao (char *str, int salto) {

        char buf[salto];  // usando buffer intermedio
        int len = strlen(str);  // pode-se resolver facilmente sem usar strlen...

        for ( int i=0; i<salto; i++ )
                buf[i] = str[i];
        for ( int i=0; i<len-salto; i++ )
                str[i] = str[i+salto];
        for ( int i=0; i<salto; i++ )
                str[len-salto+i] = buf[i];
```

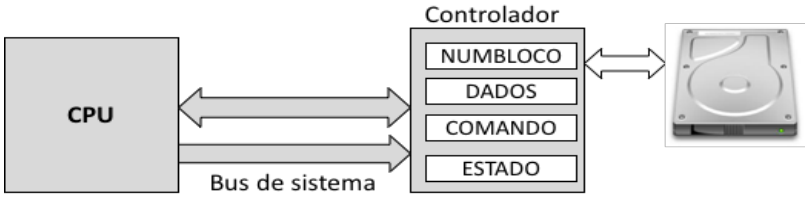
30- (1 val) Considere uma arquitetura com o controlador de discos na figura seguinte. Este permite ler ou escrever no disco blocos de 512 bytes. Neste exercício vamos considerar apenas a escrita de blocos no disco. As operações entre o CPU e o controlador são via *ports* de 16 bits a seguir descritos:

- **endereço 0x30 DADOS**: registo que, quando escrito, acrescenta os 2 bytes a um buffer interno ao controlador, do tamanho de um bloco do disco. Se efectuar uma sequência de escritas neste *port*, totalizando mais de 512 bytes, sem mandar fazer uma escrita, os bytes extra serão escritos por cima dos já existentes nesse buffer.
- **endereço 0x31 NUMBLOCO**: registo só de escrita; número do bloco do disco a ser escrito.
- **endereço 0x32 COMANDO**: registo só de escrita; quando o CPU escreve neste registo o valor 0x100, o controlador desencadeia a escrita do seu buffer interno no bloco do disco com o número indicado em NUMBLOCO.
- **endereço 0x33 ESTADO**: registo só de leitura. Depois de ser pedida a escrita de um bloco no disco (via registo COMANDO), o controlador colocará o bit 0 a 1 indicando que está ocupado; quando termina este passa a zero.

Note que não pode escrever em DADOS enquanto uma transferência para o disco está em curso, sob risco de corromper a transferência em curso.

As instruções de IN e OUT do CPU estão acessíveis em C usando as funções seguintes:

```
unsigned short in(unsigned int portid); // IN de 2 bytes
void out(unsigned int port, unsigned short data ); // OUT de 2 bytes
```



Implemente em C a função "escreveBloco" por forma a que possa mandar escrever um bloco de 512 bytes no disco, no bloco indicado. Esta função deve retornar assim que a transferência do controlador para o disco se inicia.

```
void escreveBloco( unsigned short *buff, unsigned int numBloco ) {

        #define ESTADO 0x33
        #define COMANDO 0x32
        #define NUMBLOCO 0x31
        #define DADOS  0x30

        while ( (in(ESTADO) & 1)==1 )
        ;
        for ( int i=0; i<256; i++ )
                out( DADOS, buff[i] );
        out( NUMBLOCO, numBloco );
        out( COMANDO, 0x100 );
```

Intel x86 (IA32) Assembly Language Cheat Sheet

Suffixes: b=byte (8 bits); w=word (16 bits); l=long (32 bits). Optional if instruction is unambiguous.

Operands: immediate/constant (not as *dest*): $10, $0xff ou $0b01101 (decimal, hex or bin)

32-bit registers: %eax, %ebx, %ecx, %edx, %esi, %edi, %esp, %ebp

16-bit registers: %ax, %bx, %cx, %dx, %si, %di, %sp, %bp

8-bit registers: %al, %ah, %bl, %bh, %cl, %ch, %dl, %dh

direct addr: (2000) or (0x1000+53)

indirect addr: (%eax) or 16(%esp) or 200(%edx, %ecx, 4)

Note that it is not possible for **both** *src* and *dest* to be memory addresses.

Instruction	Effect	Examples
Copying Data		
mov *src,dest*	Copy src to dest	mov $10,%eax movw %ax,(2000)
Arithmetic		
add *src,dest*	dest = dest + src	add $10, %esi
sub *src,dest*	dest = dest – src	sub %eax,%ebx
cmp *src,dest*	Compare using sub (dest is not changed)	cmp $0,%eax
inc *dest*	Increment destination	inc %eax
dec *dest*	Decrement destination	decl (0x1000)
Bitwise and Logic Operations		
and *src,dest*	dest = src & dest	and %ebx, %eax
test *src,dest*	Test bits using and (dest is not changed)	test $0xffff,%eax
or *src,dest*	dest = src | dest	or (0x2000),%eax
xor *src,dest*	dest = src ^ dest	xor $0xffffffff,%ebx
shl *count,dest*	dest = dest << count	shl $2,%eax
shr *count,dest*	dest = dest >> count	shr $4,(%eax)
sar *count,dest*	dest = dest >> count (preserving signal)	sar $4,(%eax)
Jumps		
je/jz *label*	Jump to label if dest == src /result is zero	je endloop
jne/jnz *label*	Jump to label if dest != src /result not zero	jne loopstart
jg *label*	Jump to label if dest > src	jg exit
jge *label*	Jump to label if dest >= src	jge format_disk
jl *label*	Jump to label if dest < src	jl error
jle *label*	Jump to label if dest <= src	jle finish
ja *label*	Jump to label if dest > src (unsigned)	ja exit
jae *label*	Jump to label if dest >= src (unsigned)	jae format_disk
jb *label*	Jump to label if dest < src (unsigned)	jb error
jbe *label*	Jump to label if dest <= src (unsigned)	jbe finish
jz/je *label*	Jump to label if all bits zero	jz looparound
jnz/jne *label*	Jump to label if result not zero	jnz error
jmp *label*	Unconditional jump	jmp exit
Function Calls / Stack		
call *label*	Call (Push eip and Jump)	call format_disk
ret	Return to caller (Pop eip and Jump)	ret
push *src*	Push item to stack	pushl $32
pop *dest*	Pop item from stack	pop %eax

Directives (examples):

.data – data section (global variables)

.text – text section (code)

.int – 32bits space(s) for integer value(s)

.comm *label*, *length* – length bytes space

.ascii – char sequence

.global *label* -- export *label* symbol/address

Functions Linux/32bits:

caller:

- push args (right to left)

- call function

- free stack space used with args

C types:

char 1 byte

short 2 bytes

int, float, long and *pointer* 4 bytes

double 8 bytes

callee (function):

- initialise: push %ebp
  mov %esp, %ebp
  sub $4, %esp #space for local var.

- use ebp based address, e.g.: movl 8(%ebp), %eax

- result at %eax

- finalise: mov %ebp, %esp #free local var.
  pop %ebp
  ret