Öğrenci Adı – Soyadı: _____
Öğrenci Numarası: _____

S1	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*	*Toplam*

Hacettepe Üniversitesi
Bilgisayar Mühendisliği Bölümü

BİL 220
Sistem Programlamaya Giriş

Final Sınavı

Tarih: 30 Mayıs 2012
Süre: 135 dak.

Sınava başlamadan önce aşağıda yazılanları mutlaka okuyunuz!

- Bu sınav **kapalı kaynak** bir sınavdır. Yani sınav süresince ilgili ders kitapları veya ders notlarınızdan faydalanmanız yasaktır.
- Size yardımcı olması açısından sonraki 2 sayfada bazı Intel IA32/x86-64 Assembly komutlarının söz dizimleri ve diğer bazı ilgili tanımlar verilmiştir.
- **Sınavda kopya çekmek yasaktır.** Kopya çekmeye teşebbüs edenler hakkında ilgili idare işlemler **kesinlikle** başlatılacaktır.
- Her bir sorunun toplam ağırlığı soru numarasının ardında parantez içinde belirtilmiştir.
- Sınav toplam 120 puan üzerinden değerlendirilecektir.

Sınav bu kapak sayfası dahil toplam 11 sayfadan oluşmaktadır. Lütfen kontrol ediniz!

BAŞARILAR!

Sıçrama İşlemleri

Sıçrama	Koşul
`jmp`	`1`
`je`	`ZF`
`jne`	`~ZF`
`js`	`SF`
`jns`	`~SF`
`jg`	`~(SF^OF)&~ZF`
`jge`	`~(SF^OF)`
`jl`	`(SF^OF)`
`jle`	`(SF^OF)|ZF`
`ja`	`~CF&~ZF`
`jb`	`CF`

Aritmetik İşlemler

Format		İşlem
`addl`	*Src,Dest*	Dest = Dest + Src
`subl`	*Src,Dest*	Dest = Dest - Src
`imull`	*Src,Dest*	Dest = Dest * Src
`sall`	*Src,Dest*	Dest = Dest << Src
`sarl`	*Src,Dest*	Dest = Dest >> Src
`shrl`	*Src,Dest*	Dest = Dest >> Src
`xorl`	*Src,Dest*	Dest = Dest ^ Src
`andl`	*Src,Dest*	Dest = Dest & Src
`orl`	*Src,Dest*	Dest = Dest | Src
`incl`	*Src*	Dest = Dest + 1
`decl`	*Src*	Dest = Dest - 1
`negl`	*Src*	Dest = - Dest
`notl`	*Src*	Dest = ~ Dest

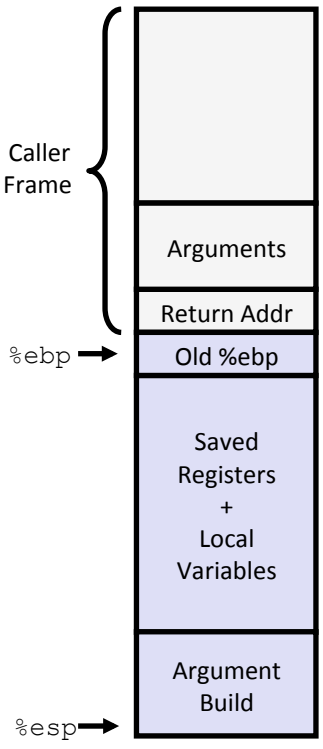
Bellek İşlemleri

Format	İşlem
(Rb, Ri)	Mem[Reg[Rb]+Reg[Ri]]
D(Rb, Ri)	Mem[Reb[Rb]+Reg[Ri]+D]
(Rb, Ri, S)	Mem[Reg[Rb]+S*Reg[Ri]]

Yazmaçlar (Registers)

63	31	15	8	7	0	
`%rax`	`%eax`	`%ax`	`%ah`	`%al`		Return value
`%rbx`	`%ebx`	`%bx`	`%bh`	`%bl`		Callee saved
`%rcx`	`%ecx`	`%cx`	`%ch`	`%cl`		Argument #4
`%rdx`	`%edx`	`%dx`	`%dh`	`%dl`		Argument #3
`%rsi`	`%esi`	`%si`		`%sil`		Argument #2
`%rdi`	`%edi`	`%di`		`%dil`		Argument #1
`%rbp`	`%ebp`	`%bp`		`%bpl`		Callee saved
`%rsp`	`%esp`	`%sp`		`%spl`		Stack Pointer
`%r8`	`%r8d`	`%r8w`		`%r8b`		Argument #5
`%r9`	`%r9d`	`%r9w`		`%r9b`		Argument #6
`%r10`	`%r10d`	`%r10w`		`%r10b`		Reserved
`%r11`	`%r11d`	`%r11w`		`%r11b`		Used for linking
`%r12`	`%r12d`	`%r12w`		`%r12b`		Callee saved
`%r13`	`%r13d`	`%r13w`		`%r13b`		Callee saved
`%r14`	`%r14d`	`%r14w`		`%r14b`		Callee saved
`%r15`	`%r15d`	`%r15w`		`%r15b`		Callee saved

Linux Yığıt (Stack) Yapısı



Özel Hizalama Durumları (Intel IA32)

1 byte: `char`, ...
sınırlandırma yok

2 bytes: `short`, ...
en düşük bit adresi 0_2

4 bytes: int, float, char *, ...
e*n düşük 2 bit adresi* 00_2

8 bytes: `double`, ...
Windows: e*n düşük 3 bit adresi* 000_2
Linux: e*n düşük 2 bit adresi* 00_2

12 bytes: `long double`
Windows & Linux:
e*n düşük 2 bit adresi* 00_2

C Veri Tipi	IA-32	X86-64
`char`	1	1
`short`	2	2
`int`	4	4
`long`	4	8
`long long`	8	8
`float`	4	4
`double`	8	8
`long double`	10/12	10/16
`pointer`	4	8

Özel Hizalama Durumları (Intel x86-64)

1 byte: `char`, ...
sınırlandırma yok

2 bytes: `short`, ...
en düşük bit adresi 0_2

4 bytes: `int`, `float`, ...
e*n düşük 2 bit adresi* 00_2

8 bytes: `double`, `char *`, ...
Windows & Linux:
e*n düşük 3 bit adresi* 000_2

16 bytes: `long double`
Linux: e*n düşük 3 bit adresi* 000_2

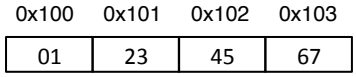
Kayan noktalı sayı (floating point)

Bias = $2^{k-1} - 1$

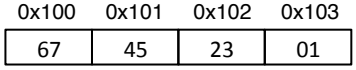
Bayt Sıralama (Byte Ordering)

`0x100` adresinde 4-bayt'lık değişken `0x01234567`

Big Endian
En anlamsız bayt en yüksek adreste



Little Endian
En anlamsız bayt en düşük adreste



Soru 1. (12 puan) *Tamsayı gösterimleri.*
6-bit'lik bir bilgisayar üzerinde,

- İşaretli tam sayılar için ikili tümler (2's complement) aritmetiği kullanılmaktadır.
- `short` tamsayılar 3-bit ile gösterilmektedir.
- Bir `short` açıkça `int`'e dönüştürülürken (cast edilirken) işaret genişletmesi (sign extension) kendiliğinden gerçekleşmektedir.
- `int`'ler üzerinde sağa kaydırma aritmetik kaydırma işlemi ile gerçekleşmektedir.

Bu varsayımlara göre aşağıdaki tanımları göz önünde bulundurarak altta verilen tablodaki boş kutucukları doldurunuz.

```
short sa = -2;
int b = 3*sa;
int a = -16;
short sb = (short) a;
unsigned ua = a;
```

İfade	Tam sayı gösterimi	İkili gösterimi
`b`	-6	111 010
`sb`	0	000
`ua`	48	110 000
`a >> 2`	-4	111 100
`ua >> 2`	12	001 100
`b << 3`	16	010 000

Soru 2. (10 puan) *Kayan noktalı sayı gösterimleri.*
Bu soruyu IEEE Standard 754 kayan noktalı sayı formatına göre oluşturulan 8-bit'lik bir kayan noktalı gösterimine göre cevaplayınız. Bu gösterimde,

- En anlamlı bit (the most significant bit) *işaret bit*'idir.
- İşaret bit'inin ardından gelen 3 bit kayan noktalı sayının *üstünü (exponent)* verir. Burada *üst için kaydırma değeri (exponent bias)* 3'tir.
- Geri kalan 4 bit ise *kesirli kısmı (fraction)* belirtir.
- Bu gösterimde ifade edilen sayılar $V = (-1)^s \times M \times 2^E$ şeklinde yazılabilen sayılardır.

(*E*: *kaydırılmış üst değeridir (biased exponent)*. *M*: *x* veya *x*/*y* şeklinde bir sayıya karşılık gelirken burada *x* bir tamsayı ve *y* ise 2'nin bir katıdır.)

Bu gösterim için belirlenen kurallar, *normalize olan sayı*, *normalize olmayan sayı*, *sonsuz* ve *NaN* icin IEEE Standard 754'e benzerdir. Buna göre aşağıda verilen tablodaki boş kutucukları doldurunuz.

NOT: "–" ile belirtilen bölümleri doldurmanıza gerek yoktur. Eğer bilgisayardaki gösterimde yuvarlama gerekiyor ise çifte-yuvarlama (round-to-even) *yapmalısınız.*

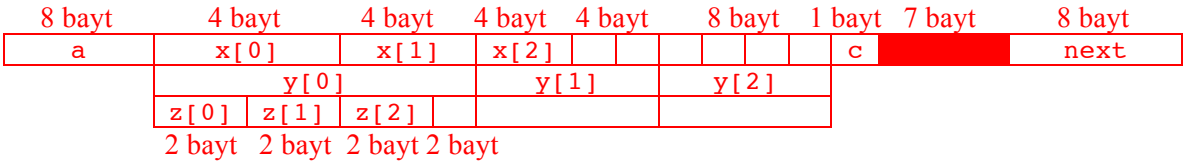
İfade	İkili değer	*M*	*E*	Değer
–	1 000 0101	5/16	-2	-5/64
–	1 110 0101	21/16	3	-10.5
Normalize olan en küçük negatif sayı	1 110 1111	31/16	3	-31/2

Soru 3. (24 puan) *Struct ve union.*
Aşağıda C programlama dilinde iki veri yapısı tanımı verilmiştir:

```
struct node {
  long a;
  union data b;
  char c;
  struct node *next;
};
```

```
union data {
  int x[3];
  long y[3];
  short z[3];
};
```

(a) (8 puan) Intel x86-64 sistemi üzerinde `struct node` veri tipinin bellekteki yerleşimini veri yapı-sındaki alanların ilgili bağıl konumlarını (offset) açıkça belirterek gösteriniz.



(b) (16 puan)Aşağıda bir grup C ve Assembly kodu verilmiştir.

```
unsigned long hawkeye(struct node *ptr) {
   return &ptr->next->b.x[1];
}

short hulk(struct node *ptr) {
   return (ptr->b.z[2]);
}

long long ironman(struct node *ptr) {
   union data *uptr =
     (union data *)&ptr->b.y[1];
    return uptr->y[1];
}

long thor(struct node *ptr) {
   return (ptr->a = (long)ptr->next);
}
```

A	`movq 40(%rdi), %rax` `addq $10, %rax`
B	`movq 40(%rdi), %rax` `movq %rax, (%rdi)`
C	`movq 40(%rdi), %rax` `addq $12, %rax`
D	`movq 24(%rdi), %rax`
E	`movswl 12(%rdi), %eax`
F	`movq 40(%rdi), %rax` `movq 8(%rax), %rax`

Linux yüklü bir Intel x86-64 sistemi üzerinde çalıştığınızı varsayarak ve yukarıdaki veri yapısı tanımlarını kullanarak her C fonksiyonunun hangi Assembly koduna karşılık geldiğini belirtiniz.

Fonksiyon	**Assembly kodu**
hawkeye	C
hulk	E
ironman	D
thor	B

Soru 4. (21 puan) *Önbellek.*
Aşağıda C programlama dilinde yazılmış üç farklı kod parçası verilmiştir. Üzerinde çalıştığınız bilgisayar sisteminin 2 küme içeren ve doğrudan eşlenen (direct mapped) bir 16 baytlık veri önbelleğe sahip olduğunu varsayınız ($E = 1$, $B = 8$, $S = 2$).

Her bir kod parçasının çalıştırıldığı ilk anda önbelleğin soğuk (cold cache) ve ayrıca `X` dizisinin önbellek ile hizalandığını yani `X[0]`'nın önbelleğin ilk kümesine yüklendiğini düşününüz. **Bütün diğer değişkenlerin yazmaçlarda (registers) saklandığını farzediniz.**

NOT: Bahsedilen önbelleği çizmeniz aşağıdaki soruları cevaplarken kolaylık sağlayacaktır!

- Kod 1:

```
float X[8], t = 0;
for(int j = 0; j < 2; j++)
  for(int i = 0; i < 8; i++)
    t += X[i];
```

(a) (5 puan) Iskalama oranı: 50%

(b) (2 puan) Bu kod parçası üzerinde çalıştığı önbellek için hangi tür yerellik (locality) barındırmaktadır? Eğer bir yerellik kullanımı söz konusu değilse bunu belirtiniz.

Uzlamsal (spatial)

- Kod 2:

```
float X[8], t = 0;
for(int j = 0; j < 2; j++)
{
  for(int i = 0; i < 7; i += 2)
    t += X[i];
  for(int i = 1; i < 8; i += 2)
    t += X[i];
  }
```

(c) (5 puan) Iskalama oranı: 100%

(d) (2 puan) Bu kod parçası üzerinde çalıştığı önbellek için hangi tür yerellik (locality) barındırmaktadır? Eğer bir yerellik kullanımı söz konusu değilse bunu belirtiniz.

Herhangi bir yerellik söz konusu değildir!

- Kod 3:

```
float X[8], float t = 0;
for(i = 0; i < 2; i++)
 for(k = 0; k < 2; k++)
   for(j = 0; j < 4; j++)
     t += X[j + i * 4];
```

(e) (5 puan) Iskalama oranı: 25%

(f) (2 puan) Bu kod parçası üzerinde çalıştığı önbellek için hangi tür yerellik (locality) barındırmaktadır? Eğer bir yerellik kullanımı söz konusu değilse bunu belirtiniz.

Hem uzlamsal (spatial) hem de zamansal (temporal)

Soru 5. (15 puan) *Kural dışı durumlarda komut akışı.*
Aşağıdaki verilen C programının üreteceği olası çıktıları nelerdir?

NOT: Bu soruyu yanıtlarken tüm fonksiyonların normal bir şekilde sonlandığını ve `printf` *fonksiyonunun arabellek kullanmadığını (unbuffered olduğunu) varsayınız.*

```
main() {
  if (fork() == 0) {
    if (fork() == 0) {
        printf("D");
    }
    else {
      pid_t pid; int status;
      if ((pid = wait(&status)) > 0) {
        printf("E");
      }
    }
  }
  else {
    if (fork() == 0) {
        printf("B");
        exit(0);
    }
    printf("C");
  }

  printf("A");

  return 0;
}
```

Aşağıdaki tabloda verilen bir çıktının olası olduğunu düşünüyorsanız ilgili sütunu *Evet,* olmadığını düşünüyorsanız *Hayır* olarak işaretleyiniz.

CADAEAB	Evet
BCDEAAA	Hayır
CDAABEA	Evet
CADEABC	Hayır
DCAAEBA	Evet

Soru 6. (12 puan) *Sistem düzeyinde girdi/çıktı.*
Aşağıda C programlama dilinde yazılmış bir kod verilmiştir.

NOT: Bu soruyu yanıtlarken tüm sistem çağrılarının başarılı olduğunu ve `read()` *fonksiyonuna yapılan çağrıların atomik olduğunu yani çalışmaya başladığında sonlanana kadar durdurulmadığını varsayınız.*

```
void read_and_print_one(int fd)
{
    char c;
    read(fd, &c, 1);
    printf("%c", c); fflush(stdout);
}

int main(int argc, char *argv[])
{
    int fd1 = open("bil220.txt", O_RDONLY);
    int fd2 = open("bil220.txt", O_RDONLY);
    read_and_print_one(fd1);
    read_and_print_one(fd2);
    if(!fork()) {
        read_and_print_one(fd1);
        read_and_print_one(fd2);
        close(fd2);                    /* ***** */
        fd2 = dup(fd1);                /* ***** */
        read_and_print_one(fd1);
        read_and_print_one(fd2);
     }

    else {
        wait(NULL);
        read_and_print_one(fd1);
        read_and_print_one(fd2);

    }

    close(fd1);
    close(fd2);
    return 0;
}
```

(a) (6 puan) `bil220.txt` metin dosyanın içeriğinin "`1234567`" olduğunu varsayarak verilen kodun çalıştırıldığında üreteceği çıktıyı yazınız.

`11223453`

(b) (6 puan) `bil220.txt` metin dosyanın içeriğini yine "`1234567`" olarak kabul ederek, yukarıda verilen koddan `*****` ile işaretlenen satırlar silinip kod çalıştırılırsa oluşacak çıktıyı belirtiniz.

`11223344`

Soru 7. (12 puan) *Kod eniyileme.*
Bir bilgisayar sisteminin işlevsel birimlerine ait performans özellikleri aşağıdaki tabloda belirtilmiştir.

İşlem	Gecikme süresi (latency)	Dağıtım süresi (issue time)
Tamsayı toplama	1	1
Tamsayı çarpma	4	1
Tamsayı bölme	36	36
Kayan noktalı sayı toplama	3	1
Kayan noktalı sayı çarpma	5	2
Kayan noktalı sayı bölme	38	38
Yükleme veya kaydetme (önbellek isabeti)	1	1

Aşağıda C programlama dilinde yazılmış iki fonksiyon verilmiştir.

1. döngü

```
int dongu1(int *a, int x, int n){
  int y = x*x;
  int i;
  for (i = 0; i < n; i++)
    x = y * a[i];
  return x*y;
}
```

2. döngü

```
int dongu2(int *a, int x, int n){
  int y = x*x;
  int i;
  for (i = 0; i < n; i++)
    x = x * a[i];
  return x*y;
}
```

Bu fonksiyonlar, yukarıda belirtilen bilgisayar sistemi üzerinde `gcc` ile derlendiklerinde içteki döngüler için aşağıdaki Assembly kodları yaratılmaktadır.

1. döngü

```
.L21:
   movl %ecx,%eax
   imull (%esi,%edx,4),%eax
   incl %edx
   cmpl %ebx,%edx
   jl .L21
```

2. döngü

```
.L27:
   imull (%esi,%edx,4),%eax
   incl %edx
   cmpl %ebx,%edx
   jl .L27
```

Bu sistemde, 1. döngü yineleme başına 3.0'lık saat çevrimine (3.0 clock cycles per iteration) ihtiyaç duyarken 2. döngü ise yineleme başına 4.0'lık saat çevrimine (4.0 clock cycles per iteration) gerek duymaktadır.

(a) (6 puan) Verilen 1. döngü, 2. döngüden sayıca bir fazla komut içermesine rağmen daha hızlı çalışmaktadır. Bunun nedenini 1-2 cümle ile kısaca açıklayınız.

1. döngüde yinelemeler arasında bir bağımlılık yoktur. Bundan dolayı 1. döngü küme komut işlemeyi (pipelining) çok daha iyi kullanmaktadır.

(b) (6 puan) `gcc` derleyicisi `-funroll-loops` parametresi ile çalıştırıldığında kodun 4-yönlü döngü yayılması (4-way loop unrolling) kullanılarak derlenmesi sağlanabilmektedir. Bu yöntem, 1. döngünün işletilmesini daha da hızlandırabilir. Bunun nedenini kısaca açıklayınız.

Bu tarz bir işlem her yinelemenin işlem yükünü (overhead) azaltacaktır.

Soru 8. (14 puan) *Sanal bellek.*
Aşağıdaki özelliklere sahip bir sistem üzerinde çalıştığınızı varsayınız:

- Sanal adreslerin uzunluğu 18 bit'dir.
- Fiziksel adreslerin uzunluğu16 bit'dir.
- Sayfa büyüklüğü 1024 bayt'dır.
- TLB, 16 elemana sahip, 4 yollu (4-way) küme birleşmeli (set associative)'dir.

TLB'in içeriği ve sayfa tablosunun (page table) ilk 32 elemanı aşağıda gösterilmiştir:

NOT: Bütün sayılar onaltılı (hexadecimal) olarak yazılmıştır.

TLB			
Index	Tag	PPN	Valid
0	05	13	0
	1E	14	1
	10	0F	1
	0F	1E	0
1	1F	01	1
	11	1F	0
	03	2B	1
	1D	23	0
2	06	08	1
	0F	19	1
	0A	09	1
	1F	20	1
3	03	13	0
	13	12	1
	0C	0B	0
	2E	24	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	2D	0
05	13	1	15	1B	0
06	0F	1	16	0C	0
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	01	0	1A	0C	1
0B	16	1	1B	12	1
0C	01	1	1C	1E	0
0D	15	1	1D	0E	1
0E	0C	0	1E	27	1
0F	14	0	1F	18	1

(a) (3 puan) Aşağıdaki diyagram bir sanal adresin formatını göstermektedir. Bu diyagram üzerinde hangi bit'lerin aşağıda listelenen hangi alanlarla ilişkili olduğunu gösteriniz.

O: Sanal sayfa ötelemesi (virtual page **offset**)
N: Sanal sayfa numarası (virtual page **number**)
I: TLB indisi (TLB **index**)
T: TLB künyesi (TLB **tag**)

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T/N	T/N	T/N	T/N	T/N	T/N	I/N	I/N	O	O	O	O	O	O	O	O	O	O

(b) (3 puan) Aşağıdaki diyagram bir fiziksel adresin formatını göstermektedir. Bu diyagram üzerinde hangi bit'lerin aşağıda listelenen hangi alanlarla ilişkili olduğunu gösteriniz.

O: Fiziksel sayfa ötelemesi (physical page **offset**)
N: Fiziksel sayfa numarası (physical page **number**)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N	N	N	N	N	N	O	O	O	O	O	O	O	O	O	O

Aşağıda verilen sanal adresler için erişilen TLB elemanlarını (TLB entries) ve fiziksel adresleri yazınız. TLB'nin ıskalayıp ıskalamadığını ve bir sayfa hatası (page fault) oluşup oluşmadığını belirtiniz. Eğer bir sayfa hatası oluşuyorsa, "PPN" için "-" yazınız ve fiziksel adresi boş bırakınız.

(c) (4 puan) Sanal Adres: `0x0718F`

Sanal adres (her kutucuk 1 bit)

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1	1	1

Adres çevrimi

Parametre	*Değer*	*Parametre*	*Değer*
VPN	`0x1C`	TLB başarılı? (E/H)	Hayır
TLB İndisi	`0x0`	Sayfa hatası? (E/H)	Evet
TLB Künyesi	`0x07`	PPN	`0x-`

Fiziksel adres (her kutucuk 1 bit)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

(d) (4 puan) Sanal Adres: `0x04AA4`

Sanal adres (her kutucuk 1 bit)

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	0

Adres çevrimi

Parametre	*Değer*	*Parametre*	*Değer*
VPN	`0x12`	TLB Başarılı? (E/H)	Hayır
TLB İndisi	`0x2`	Sayfa Hatası? (E/H)	Hayır
TLB Künyesi	`0x04`	PPN	`0x0E`

Fiziksel adres (her kutucuk 1 bit)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	1	0	1	0	0	1	0	0