Arhitectura Sistemelor de Calcul (ASC)
Examinarea finală
Varianta 1
(2023 - 2024)

Anul I, Semestrul I
9 februarie 2024
Cristian Rusu

Nume: ______________________________

Prenume: ______________________________

Grupa: ______________________________

Completați aici totul cu majuscule.

Toate răspunsurile sunt în albastru.

Înainte de a începe, citiți cu atenție indicațiile următoare:

- Testul și rezolvarea sa vor fi disponbile online în zilele următoare.
- *Nu aveți voie cu laptop-uri sau alte dispozitive de comunicație.*
- *Nici calculatoarele de buzunar nu sunt permise.*
- *Vă rugăm să vă opriți telefoanele mobile.*
- Pentru întrebările cu răspunsuri multiple/simple folosiți tabelele puse la dispoziție.
- Acest test are 6 enunțuri totalizând 100 de puncte.
- Aveți la dispoziție 120 de minute pentru a completa examinarea.
- Mult succes!

Întrebarea 1. (22 puncte)

Completați tabelul de mai jos cu valorile numerice corecte. Toate numerele sunt naturale pe 12 biți. (Fiecare răspuns corect valorează 1 punct)

binar	octal	zecimal	hexazecimal
b000111100110	o0746	486	0x1E6
b000100111001	o0471	313	0x139
b000100011010	o0432	282	0x11A
b000010101010	o0252	170	0x0AA

a binar	a hexa	b binar	b hexa	a+b zecimal	a+b binar	a+b hexa
b000100011111	0x11F	b000101001001	0x149	616	b001001101000	0x268

a binar	a hexa	b binar	b hexa	axb zecimal	axb binar	axb hexa
b000101011001	0x159	b000011110110	0x0F6	84870	0b10100101110000110	0x14B86

Răspunsurile care țin cont de overflow sunt de asemenea corecte.

Întrebarea 2. (13 puncte)

Completați tabelul de mai jos cu valorile numerice corecte. Toate numerele sunt întregi pe 12 biți. (Fiecare răspuns corect valorează 1 punct)

binar	octal	zecimal	hexazecimal
b110111110100	o6764	-524	0xDF4
b110011101100	o6354	-788	0xCEC

a zecimal	a binar	a hexa	b zecimal	b binar	b hexa
-1401	b101010000111	0xA87	-15	b111111110001	0xFF1

produs axb zecimal	produs axb binar	produs axb hexa
21015	b101001000010111	0x5217

Răspunsurile care țin cont de overflow sunt de asemenea corecte.

3.1.	Grace Hopper
3.3.	Margaret Hamilton
3.5.	$1970 + 2^{31}/2^{25} = 2034$
3.7.	$\max(n, m) + 1$
3.9.	$2^8 = 256$
3.11.	10
3.13.	overflow, 420 pe 8 biți este 164
3.15.	AND
3.17.	$\frac{1}{0.8+\frac{0.2}{2}} = 1.11$
3.19.	1.67
3.21.	a XOR b
3.23.	ab
3.25.	$a\&(m-1)$
3.27.	$(a + (2^N - 1) \times b)\&(2^N - 1)$ sau $(a + \text{not}(b) + 1)\&(2^N - 1)$
3.29.	WAW
3.31.	un editor hexa (binare)
3.33.	0xAA55

Întrebarea 3. (17 puncte)

Răspundeți la următoarele întrebări scurte. Completați în tabelul de pe pagina anterioară.

3.1. Cine a scris primul compilator COBOL?

3.3. Cine a scris o mare pare din secvența de cod Assembly folosită pentru misiunea Apollo?

3.5. Folosim 32 de biți ca să numărăm câte secunde au trecut de la 1 ianurie 1970 (UNIX time). Presupunând că un bit din cei 32 este rezervat pentru semn, în ce an vom avea overflow? (presupunem că într-un an sunt 31556926 $\approx 2^{25}$ secunde)

3.7. Avem două numere naturale: a pe n biți și b pe m biți. Pe câți biți este suma $a + b$?

3.9. Când măsurăm entropia utilizând logaritmul în baza 2 atunci răspunsul calculat este în biți. În ce bază logaritmică ar trebui să facem calculul ca rezultatul să fie în bytes?

3.11. Care este reprezentarea numărului natural B în baza B pentru $B > 1$?

3.13. Avem o secvență de cod care ar trebui să returneze valoarea 420. Dar când verificăm valoarea, găsim 164. Ce se întâmplă?

3.15. Avem două numere naturale a, b pe n biți. Avem relația $a + b - (a \text{ OR } b) = a$??? b. Ce operație logică puneți în loc de ??? care balansează ecuația?

3.17. Avem un sistem de calcul cu o unitate Floating-Point Unit (FPU). Această unitate este îmbunătățită la timpul de execuție de două ori. Avem un program pentru care 20% din operații sunt FP. Atunci îmbunătățirea vitezei (speed-up) programului este:

3.19. Avem o secvență de cod care este paralelizabilă în proporție de 40%. Presupunând că avem la dispoziție oricâte core-uri de procesare s, care este îmbunătățirea maximă posibilă?

3.21. Fie a, b și c variabile digitale/binare. Simplificați maxim expresia logică $(a + b) \times (!a + !b)$

3.23. Fie a, b și c variabile digitale/binare. Simplificați maxim expresia logică $(a + b) \times (b + c) \times (a + !b) \times (b + !c)$

3.25. Vi se dă un număr natural a. Trebuie să calculați a mod m și știm despre m că este o putere a lui 2, adică $m = 2^M$. Cum faceți? Simplificați maxim expresia logică.

3.27. Vi se dau două numere naturale a și b pe N biți. Avem nevoie să calculăm $a - b$ dar sistemul nostru de calcul nu are implementată reprezentarea numerele întregi (complement față de doi) și știe să facă doar operațiile aritmetice: $+$, $\times$, div și mod. Operațiile logice sunt toate implementate. Presupunem că rezultatul operației $a - b$ este tot un număr natural. Explicați cum faceți scăderea pentru a, b și N generale în aceste condiții?

3.29. Pe un sistem de calcul avem regiștrii R1,...,R9. O secvență de cod assembly are două instrucțiuni R2 <- R4 + R7 și R2 <- R1 + R3. Ce fel de hazard este acesta?

3.31. HxD este ...

3.33. Care este numărul magic pentru un bootloader valid?

Întrebarea 4. (16 puncte)

Vrem să realizăm operația $a+b$ pentru două numere a și b pe N biți. La curs/seminar am văzut că trebuie să știm ce fel de numere sunt a și b și am discutat de două posibități: fie ambele sunt naturale, fie ambele sunt întregi (cu reprezentare în complement față de doi). Ce se întâmplă dacă a este întreg (complement față de doi) și b este natural? Răspundeți la următoarele cerințe:

4.1. (3 puncte) Dacă $N = 3$ biți, care sunt valorile min/max pentru a, b și $a + b$?

4.2. (3 puncte) Pentru un N general, care sunt valorile min/max pentru a, b și $a+b$? Pe câți biți este $a + b$?

4.3. (10 puncte) Explicați cum se face adunarea binară în acest caz. Discutați cazurile în funcție de MSB ale numerelor a și b. Explicați intervalele rezultatului în fiecare caz. (nu trebuie să desenați circuit, doar o metodă explicată în principiu de cum calculați și cum interpretați rezultatul)

4.1. $-4 \leq a \leq 3$, $0 \leq b \leq 7$ și $-4 \leq a + b \leq 10$.

4.2. $-2^{N-1} \leq a \leq 2^{N-1} - 1$, $0 \leq b \leq 2^N - 1$ și $-2^{N-1} \leq a + b \leq 2^N + 2^{N-1} - 2$. Rezultatul $a + b$ este pe $N + 1$ biți. Se acceptă și răspunsul $N + 2$ biți: $N + 1$ ca înainte plus un bit ca să știm sigur dacă rezultatul pe $N+1$ biți este natural sau reprezentat în complement față de doi.

4.3. Notăm cu a_i al i-lea bit din a și analog b_i pentru b. Atunci cele două numere sunt: $a = \sum_{i=0}^{N-2} a_i 2^i - a_{N-1}2^{N-1}$, $b = \sum_{i=0}^{N-1} b_i 2^i = \sum_{i=0}^{N-2} b_i 2^i + b_{N-1}2^{N-1}$. Scriem așa separat pentru bitul $N - 1$ pentru că acolo este diferența dintre natural și complement față de doi. Observăm că:

1. Dacă $a_{N-1} = b_{N-1} = 0$ atunci $s_i = a_i + b_i$, $i = 1, \ldots, N - 1$, deci adunare standard (a două numere naturale unul pe N biți și celălalt pe $N - 1$ biți). Rezultatul este pozitiv în intervalul $[0, 2^N - 2]$.

2. Dacă $a_{N-1} = b_{N-1} = 1$ atunci avem $s_i = a_i + b_i$, $i = 1, \ldots, N - 2$ adunare standard cu ambele numere pe $N - 1$ biți (puterea negativă din complement față de doi se anulează cu aceeași putere pozitivă din numărul natural). Rezultatul este pozitiv în intervalul $[0, 2^N - 2]$.

3. Dacă $a_{N-1} = 0$ și $b_{N-1} = 1$ atunci avem adunarea dintre un număr pe N biți și un număr pe $N - 1$ biți ambele pozitive. Rezultatul este pozitiv în intervalul $[2^{N-1}, 2^N + 2^{N-1} - 2]$

4. Dacă $a_{N-1} = 1$ și $b_{N-1} = 0$ atunci avem un număr întreg în complement față de doi. Rezultatul este întreg în intervalul $[-2^{N-1}, 2^N - 1]$.

Întrebarea 5. (14 puncte)

Vi se dă un număr a pe 32 de biți în format IEEE FP. Răspundeți la următoarele întrebări:

5.1. (4 puncte) Scrieți secvențe scurte aritmetice/logice pentru a extrage din a variabilele *semn*, *exponent* și *mantisa* care să conțină doar biții marcați cu aceste semnificații din a.

5.2. (6 puncte) Scrieți o secvență de pseudo-cod care calculeaza int(a) - o funcție care returnează partea întreagă a variabilei float a. Puteți să folosiți cele 3 variabile calculate la punctul anterior.

5.3. (4 puncte) Vi se dă un număr scris în format *floating point* în care mantisa are M biți. Care este numărul maxim de zecimale (în baza 10) pe care putem să îl avem în acest caz?

5.1. semn = $a \gg 31$, exponent = $(a \gg 23)$ & 0x000000FF, mantisa = a & 0x007FFFFF.

5.2. secvența este următoarea:
E = exponent - 127
dacă (E < 0) atunci return 0
dacă (E > 30) atunci return overflow

dacă (E < 23) atunci
x = (1 ≪ E) | (mantisa ≫ (23 - E))
altfel
x = (1 ≪ E) | (mantisa ≪ (E - 23))

dacă (semn este 1) atunci x = not(x) + 1

return x

5.3. $\lfloor \log_{10} 2 \times M \rfloor \approx \frac{M}{3}$

Întrebarea 6. (18 puncte)

La Seminarul 0x00 am discutat despre algoritmul de căutare binară. Răspundeți la următoarele cerințe:

6.1. (3 puncte) Am văzut că de multe ori compilatoarele nu preferă cod recursiv. Modificați codul de la Seminarul 0x00 astfel încât să înlăturați recursivitatea.

6.2. (3 puncte) Câte salturi sunt în codul de la punctul 6.1? Preziceți salturile sau incercați să le înlăturați. Explicați unde puteți să faceți asta și unde nu.

6.3. (6 puncte) Aveți secvența de cod de mai jos. Este aceasta echivalentă cu căutarea binară? Cum diferă? Pentru noua secvență de cod, eliminați saltul cu o expresie aritmetică/logică. La compilare ce instrucțiune va fi folosită?

6.4. (6 puncte) Algorimul de la punctul 6.3. accesează eficient memoria? Dacă da, când? Modificați algoritmul astfel încât să acceseze cât mai eficient memoria.

```
int *base = arr, len = n;
while (len > 1) {
    int mid = len/2;
    if (base[mid - 1] < x) base = base + mid;
    len = len - mid;
}
if (*base == x) return *base;
else return NOT_FOUND;
```

6.1. Căutare binară nerecursiv

```
int left = 0; right = n - 1;
while (l < r) {
        int mid = l + (r-l)/2;
        if (arr[mid] > x) r = mid-1;
        else l = mid + 1;
}
if (arr[mid] == x) return arr[mid];
else return NOT_FOUND;
```

6.2. Sunt două instrucțiuni de salt: `while (l < r)` predicție sare mereu (greșim o singură dată), `if (arr[mid] > x)` predicția aici nu poate fi făcută cu precizie mare

6.3. Secvența este echivalentă cu căutare binară (dar în loc să avem stânga/dreapta avem baza și lungimea secțiunii de vector unde căutăm), dar va face mereu $\lceil \log_2 n \rceil$ iterații. Saltul `while (l < r)` este acum exact predictibil (eroare zero). Dacă `len = len - mid` este pe else am face mai puține iterații, dar nu am fi putut elimina saltul: codul este `base = base + (base[mid - 1] < x)*mid`, iar instrucțiunea va fi un `cmov` (*conditional move*).

6.4. Problema este distanța dintre pivoți. Spre sfârșitul algoritmului pivoții sunt apropiați, dar în rest (și mai ales la început) distanțele în memorie dintre pivoți sunt mari. Soluția: în loc să avem vectorul sortat crescător/descrescator vom avea numerele în ordinea indecșilor pivoților. Ordinea ar trebui să fie ordinea BFS pe un arbore binar de căutare (se numește ordine `eytzinger`) iar codul se reduce la `while (k <= n) k = 2*k + (arr[k] < x)`.
Pentru detalii: `https://cglab.ca/~morin/misc/arraylayout/`