Arhitectura Sistemelor de Calcul (ASC)
Examinarea finală
Varianta 2
(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
b000110101100	o0654	428	0x1AC
b000101000111	o0507	327	0x147
b000010000011	o0203	131	0x083
b000010010101	o0225	149	0x095

a binar	a hexa	b binar	b hexa	a+b zecimal	a+b binar	a+b hexa
b000011100011	0x0E3	b000000101011	0x02B	270	b000100001110	0x10E

a binar	a hexa	b binar	b hexa	axb zecimal	axb binar	axb hexa
b000110110011	0x1B3	b000110011100	0x19C	179220	0b101011110000010100	0x2BC14

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
b111011000101	o7305	-315	0xEC5
b101111010010	o5722	-1070	0xBD2

a zecimal	a binar	a hexa	b zecimal	b binar	b hexa
-1049	b101111100111	0xBE7	-24	b111111101000	0xFE8

produs axb zecimal	produs axb binar	produs axb hexa
25176	b110001001011000	0x6258

Răspunsurile care țin cont de overflow sunt de asemenea corecte.

3.2.	Claude Shannon
3.4.	John von Neumann
3.6.	$1970 + 2^{32}/2^{25} = 2098$
3.8.	$n + m$
3.10.	$2^4 = 16$
3.12.	100
3.14.	overflow, 425 pe 8 biți este 169
3.16.	OR
3.18.	$\frac{1}{0.7+\frac{0.2}{2}+0.1\times\frac{3}{2}} = 1.05$
3.20.	1.25
3.22.	$a+!b$
3.24.	$a(b + c)$
3.26.	$a\&(m - 1)$
3.28.	$(a + (2^N - 1) \times b)\&(2^N - 1)$ sau $(a + \text{not}(b) + 1)\&(2^N - 1)$
3.30.	WAR
3.32.	un compresor de binare executabile
3.34.	Power-On Self Test (POST)

Întrebarea 3. (17 puncte)

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

3.2. Cine e considerat părintele teoriei informației?

3.4. Matematician de origine maghiară care are o arhitectură de calcul care îi poartă numele

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

3.8. Avem două numere naturale: a pe n biți și b pe m biți. Pe câți biți este produsul $a \times b$?

3.10. 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 nibbles?

3.12. Care este reprezentarea numărului natural B^2 în baza B pentru $B > 1$?

3.14. Avem o secvență de cod care ar trebui să returneze valoarea 425. Dar când verificăm valoarea, găsim 169. Ce se întâmplă?

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

3.18. 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. Dar această îmbunătățire a vitezei pentru FP vine cu un dezavantaj, pentru alte 10% operații pentru accesul memorie viteza scade cu un factor de 2/3. Atunci îmbunătățirea vitezei (speed-up) programului este:

3.20. Avem o secvență de cod în care partea paralelizabilă este în proporție de trei ori mai scurtă decât cea secvențială. Presupunând că avem la dispoziție $s = 5$ core-uri de procesare, care este îmbunătățirea maximă posibilă?

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

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

3.26. 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.

3.28. 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?

3.30. Pe un sistem de calcul avem regiștrii R1,...,R9. O secvență de cod assembly are două instrucțiuni R4 <- R1 + R5 și R5 <- R1 + R2. Ce fel de hazard este acesta?

3.32. UPX este ...

3.34. Cum se numește secvența de teste pe care o face calculatorul vostru la boot pentru a se asigura că hardware-ul funcționează corect?

Întrebarea 4. (8 puncte)

La Seminarul 0x00 am discutat despre două versiuni ale algoritmului de căutare binară pentru un vector de dimensiune N. Considerăm că dimensiunea N este un număr natural pe 32 de biți. Răspundeți la următoarele întrebări:

4.1. (2 puncte) În cazul algoritmului *binarySearch2*, care este valoarea maximă posibilă pentru variabila *mid*?

4.2. (6 puncte) Amintiți-vă diferența dintre *binarySearch1* și *binarySearch2*, o variantă este mai rapidă (număr de operații) dar cealaltă este corectă (nu suferă de overflow). Presupunând că aveți disponibile ambele implementări, combinați cei doi algoritmi (adică puteți face apeluri multiple la amebele funcții) astfel încât procedura rezultată să fie și corectă dar și eficientă.

4.1. Valoarea maximă pentru *mid* este N ($2^{32}-1$ pentru valoarea maximă a lui N pe 32 de biți).

4.2. Vom nota cu MAXN $= 2^{32} - 1$ valoarea maximă pe care N o poate lua.
Pseudo-codul este:
DACĂ $N \leq$ MAXN/2 ATUNCI apel *binarySearch2()*
ALTFEL apel *binarySearch1()*
Mai sus, /2 se face păstrând calculele în domeniul numerelor naturale (operație făcută cu deplasare). În cazul nostru: MAXN/2 $= 2^{31} - 1$.

Întrebarea 5. (18 puncte)

Se dă un număr natural $a \neq 0$ pe 16 biți. Notăm cu a_i al i-lea bit din a (avem $i = 0, \ldots, 15$). Răspundeți la următoarele întrebări și realizați următoarele cerințe:

5.1. (7 puncte) Găsiți o secvență de operații aritmetice și logice prin care să convertiți numărul a în sistemul de reprezentare în virgulă mobilă IEEE 754 Floating Point pe 32 de biți? Dați secvențe de operații aritmetice/logice pentru bitul de semn, exponent și mantisă.

5.2. (3 puncte) Se pot converti toate numerele naturale $a \neq 0$ pe 16 biți în formatul IEEE FP pe 32 de biți? Dar dacă $a \neq 0$ natural este pe 32 de biți? Motivați răspunsurile.

5.3. (5 puncte) Care este cel mai mic număr natural (indiferent de numărul de biți) care nu se poate reprezenta în formatul IEEE FP pe 32 de biți? Dați reprezentarea acestui număr în hexazecimal.

5.4. (3 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. Toate numerele a sunt naturale, deci bitul de semn $s = 0$. Pentru mantisă și exponent trebuie întâi să aflăm $i_{\max}$ care este poziția primului bit "1" în reprezentarea binară a lui a. Atunci avem $\textit{exponent} = i_{\max} + 127$ (avem +127 este pentru că apoi în formatul IEEE FP scădem 127 la exponent) iar biții de la mantisă sunt $\textit{mantisa} = a_{i_{\max}-1}, a_{i_{\max}-2}, \ldots, a_0$ (pentru că în formatul IEEE FP avem deja "1." după care vine mantisa, dar primul "1" e deja pus în reprezentare). Numărul a va pune maxim 15 biți în mantisă (iar în mantisă ar încăpea 23 de biți, deci este spațiu suficient). Acest $i_{\max}$ apare calculat în Seminarul 0x00, este $\lfloor \log_2 a \rfloor$.

5.2. Cele pe 16 biți da, cele pe 32 de biți nu. Pentru că sunt doar 23 de biți în mantisă.

5.3. 16777217 zecimal și 0x4B800000 hexazecimal, înainte este 16777216 (acesta este 2^{24}) iar după vine 16777218. Indiferent ce e mantinsa, pentru că exponentul lui 2 este 24 atunci ultimul bit din reprezentare (al 24-lea) este zero deci numerele nu mai pot fi impare (pentru că ultimul bit reprezintă $2^0 = 1$ în zecimal iar mantisa are doar 23 de biți). Valoarea 24 se obține punând 151 în exponent în IEEE FP 32 de biți (astfel încât $151 - 127 = 24$). Acesta este exemplul dat și la curs.

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

Întrebarea 6. (22 puncte)

Am discutat la curs că sistemele de calcul moderne folosesc o mulțime de tehnici (printre care *out-of-order execution*, *branch prediction*, *caching*). Răspundeți la următoarele întrebări:

6.1. (4 puncte) Avem regiștrii R1,...,R9 și o secvență assembly:

I1. R1 <- locație memorie 1

I2. R2 <- locație memorie 2

I3. R3 <- locație memorie 3

I4. R2 <- R2 + R3

I5. R1 <- R1 + R2 + R3

instrucțiunea I1 se încheie în al 5lea ciclu de ceas, I2 în al 2lea ciclu de ceas și I3 în al 3lea ciclu de ceas. La care ciclu de ceas se calculează I4 și I5 dacă sistemul de calcul are *out-of-order execution*?

6.2. (6 puncte) Sistemul de calcul are și memorie cache (L1/L2/L3). Blocurile de cache sunt de 64 de bytes și nu pot fi controlate direct de către programator (adică, la citirea unui byte din memorie blocul de cache ia și următoarele 63 de valori). În programul nostru, citim un byte din memorie. Cum ne putem da seama dacă byte-ul respectiv vine din memoria principală RAM sau e din cache?

6.3. (6 puncte) Sistemul de calcul are și un sistem de *branch prediction*: dacă un salt s-a realizat mereu în ultimele 7 situații atunci predicția este de salt la pasul următor. Aveți funcția *citește_valoare* mai jos care returnează elementul de la poziția `index` dintr-un `vector`. Folosiți și informația de la punctul 6.2. și scrieți o secvență de cod cu cât mai puține salturi condiționale care aduce în memoria cache rând pe rând câte un element de după dimensiunea maximă a vectorului.

6.4. (6 puncte) Folosind 6.1.-6.3., explicați cum putem în principiu să extragem din cache informația de după dimensiunea maximă a vectorului?

```
funcție citește_valoare(index):
     dacă (index < max_dimensiune_vector) atunci returnează vector[index]
     altfel returnează NULL
```

6.1. Ciclurile de ceas 4 și 6.
6.2. Măsurăm timpul, este singura diferență între accesul la memoria RAM principală versus memoria cache L1. Folosim funcții pentru a măsura timpul înainte și după citirea din memorie.
6.3. Folosim 6.1., adică `vector[index]` e accesat înainte să avem condiția de la `dacă` evaluată: pentru i=1,...
index = i+(max_dimensiune_vector + j - i)*!(i & 7); j = j + !(i & 7)
valoare = citește_valoare(index)
6.4 Acum că avem `vector[index]` în cache trebuie să îl folosim cumva. O secvență de cod `buffer[vector[index]*512]` (putea să fie orice aici, nu doar 512 dar ideea este să fie mult mai mare decât 64 - dimensiunea unui bloc de cache). Și acum folosim idea de la 6.2., dacă `buffer[5*512]` se citește rapid atunci la `vector[index]` a fost valoarea 5. Deci, la un `index` peste `max_dimensiune_vector` se află valoarea 5. Se încearcă sistematic valorile 0-255 (un byte).

Acesta e atacul Spectre simplificat, pentru detalii: `https://spectreattack.com/spectre.pdf`