Microprocessoren
Studieleidraad

Coördinerend verantwoordelijke:

Prof. dr. Eddy Flerackers
eddy.flerackers@uhasselt.be

Cotitularis:

Prof. dr. Frank Van Reeth
frank.vanreeth@uhasselt.be

Andere leden van het onderwijsteam:

Patrik Goorts
patrik.goorts@uhasselt.be
Raf Menten
raf.menten@uhasselt.be
Sammy Rogmans
sammy.rogmans@uhasselt.be

Academiejaar 2013–2014

Inhoudsopgave

Hoofdstuk 1

Inleiding studieleidraad

1.1 Doelstellingen

Voordbouwend op de kennis opgedaan in opleidingsonderdeel *inleiding tot computer- en communicatiesystemen* verwerft de student een basiskennis over microprocessoren en programmatie in assembler.

1.2 Voorkennis

Inleiding tot computer- en communicatiesystemen (1144)

1.3 Didactische werkvormen

Zelfstudieopdrachten, een project en/of oefeningen vormen de basis van dit opleidingsonderdeel. Hoorcolleges worden gebruikt voor het situeren van de leerstof in een breder kader en om belangrijke/moeilijke onderdelen te doceren. Waar nodig vormen responsiecolleges een aanvulling bij de begeleiding van de zelfstudie.

1.4 Studiemateriaal

Deze cursustekst bestaat uit 2 delen en biedt je de mogelijkheid om de leerinhoud te bestuderen, en kan tevens gebruikt worden als een naslagwerk. Een studieleidraad gidst je door de cursustekst en helpt je de leerinhoud te verwerken met behulp van zelfstudie-opdrachten. Deze leidraad is online beschikbaar en zal regelmatig aangevuld worden.

1.5 Evaluatie

De evaluatie bestaat uit een schriftelijk examen.

1.6 Structuur van de studieleidraad

In deze studieleidraad komen opdrachten van verschillende soorten voor. Telkens wordt een symbool gebruikt dat het soort opdrachten kenmerkt:

- Leesopdrachten, waarbij je gevraagd wordt een gedeelte van het cursusmateriaal te bestuderen.

- Oefeningen, waarbij je het bestudeerde materiaal toepast in opdrachten "op papier".

- Programmeeroefeningen, waarbij je het bestudeerde materiaal toepast.

- Denkoefeningen, die je doen nadenken over het gelezen cursusmateriaal.

- Extra oefeningen die aanvullende opdrachten bevatten die gemaakt kunnen worden in de eventuele resterende tijd.

Hoofdstuk 2

Herhaling: Voorstelling van gegevens

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- de belangrijkste talstelsels voor computersystemen kent.
- kan werken met deze talstelsels.
- de belangrijkste voorstellingswijzen voor gehele getallen kent en begrijpt
- de belangrijkste voorstellingswijzen voor karakters kent en begrijpt

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **1,5** uren in beslag nemen.

2.1 Talstelsels

Belangrijke talstelsels in de informatica zijn

- tweetallig stelsel (binair)
- tientallig stelsel (decimaal)
- zestientallig stelsel (hexadecimaal)

In het tientallig stelsel krijgt elk cijfer in een getal een bepaald 'gewicht', we noemen dit eenheden, tientallen, honderdtallen, ... (zie tabel 2.1)
Zo ook in het binair talstelsel (tabel 2.2):

10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}
7	6	5	3	.	2	1	5
↓	↓	↓	↓		↓	↓	↓
$\times 1000$	$\times 100$	$\times 10$	$\times 1$		$\times 1/10$	$\times 1/100$	$\times 1/1000$

Tabel 2.1: Werking van het tientallig stelsel.

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
1	1	0	1	.	1	0	1
↓	↓	↓	↓		↓	↓	↓
$\times 8$	$\times 4$	$\times 2$	$\times 1$		$\times 1/2$	$\times 1/4$	$\times 1/8$

Tabel 2.2: Werking van het binair stelsel.

Dit binair getal schrijven we tiendelig als:

$$\begin{aligned} 1\times 8+1\times 4+0\times 2+1\times 1+1\times 1/2+0\times 1/4+1\times 1/8 &= 8+4+1+0.5+0.125 \\ &= 13.625 \end{aligned}$$

Om een tiendelig getal (vb 28.7_{10}) om te zetten in binair gaan we te werk zoals geïllustreerd in tabel 2.3.

28.7	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$		$2^{-1}=0.5$	$2^{-2}=0.25$	...
-16	1	1	1	0	0	.	1	0	...
12.7									
-8.0									
4.7									
-4.0									
0.7									
-0.5									
0.2									
-0.125									
0.075									
⋮									

Tabel 2.3: Voorbeeld: Tiendelig getal omzetten in binair: $28.7_{10} \rightarrow$ binair.

Het tiendelig getal 28.7 kan dus binair benaderd worden door 11100.101. Het is duidelijk dat een 'exact' tiendelig getal niet altijd 'exact' binair weer te geven is.

Een binair cijfer (1 of 0) noemt men een bit (= binary digit). Een reeks van vier bits noemt men een nibble. Een reeks van acht bits noemt men een byte. Een of

meer bytes noemt men een woord (het aantal bytes in een woord is afhankelijk van het computersysteem). De meest linkse bit in een binair getal noemt men de *MSB*, de meest rechtse bit is de *LSB*. Dit wordt getoond in tabel 2.4.

1	0	1	1	0	1	1
↓						↓
MSB						LSB

Tabel 2.4: De most significant bit en least significant bit van een binair getal.

In het binair talstelsel worden de getallen al snel groot (vergelijk bv. 20_{10} met 10100_2) en weinig overzichtelijk. Daarom voert men het hexadecimaal talstelsel in om de bits per 4 te groeperen. Met 4 bits kan men tot 15 (1111_2) met 1 cijfer weergeven; daarvoor zijn extra symbolen nodig (zie tabel 2.5).

tiendelig:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
hexadecimaal:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Tabel 2.5: Hexadecimale symbolen.

De omzetting van een hexadecimaal getal naar een decimaal getal gebeurt zoals getoond in tabel 2.6.

16^3	16^2	16^1	16^0		16^{-1}	16^{-2}
1	D	6	E	.	2	3
↓	↓	↓	↓		↓	↓
$\times 4096$	$\times 256$	$\times 16$	$\times 1$		$\times 1/16$	$\times 1/256$

Tabel 2.6: Voorbeeld: Omzetting van een hexadecimaal getal naar een decimaal getal.

Dus:

$$\begin{aligned} 1D6E.23_{16} &= 1 \times 4096 + 13 \times 256 + 6 \times 16 + 14 \times 1 + 2 \times 0.0625 + 3 \times 0.00390625 \\ &= 7534.06640625_{10} \end{aligned}$$

Hexadecimale getallen worden dikwijls voorafgegaan met 0x i.p.v. een index 16:

$$0x5E6C$$

Een binair getal kan omgezet worden in een hexadecimaal getal door te groeperen per 4 bits:

$$\begin{aligned} 110101101110 &= 1101\ 0110\ 1110 \\ &= 0xD6E \end{aligned}$$

Een byte (8 bit!) wordt voorgesteld door 2 hexadecimale cijfers.

2.2 Coderen van gegevens

2.2.1 Coderen van integers

Gehele getallen of integers kunnen op verschillende manieren in binaire vorm worden voorgesteld. Een mogelijkheid is het tiendelig getal gewoon om te zetten in het binair talstelsel.
Wat echter gedaan indien het getal negatief is?
Hoe kan men het teken in binaire code schrijven?

Doorgaans gebruikt men voor weergave van het teken een van volgende drie mogelijkheden:

- sign-magnitude
- twos-complement
- ones-complement

De meest eenvoudige voorstelling biedt de sign-magnitude methode: als de *MSB* gelijk is aan 0 dan is het getal positief, als de *MSB* gelijk is aan 1 dan is het getal negatief. Deze bit noemt men dan ook de tekenbit. Merk op dat we in deze code twee vormen hebben voor het getal nul: 0000 en 1000.

Om met de twos-complement methode een negatief getal voor te stellen, gaat men als volgt te werk: men zet de absolute waarde om in een binair getal, verandert elke 1 in een 0 en elke 0 in een 1 (complement nemen) en telt 1 bij (zie tabel 2.7).

		compl		+1		
-6	→	0110	→	1001	→	1010

Tabel 2.7: Voorbeeld: Twos-complement methode om een negatief getal voor te stellen.

Bij de ones-complement methode wordt hetzelfde algoritme gebruikt behalve de laatste stap: er wordt geen 1 bijgeteld (zie tabel 2.8).

		compl		
-6	→	0110	→	1001

Tabel 2.8: Voorbeeld: Ones-complement methode om een negatief getal voor te stellen.

De complement methodes zijn rekentechnisch interessanter dan de sign-magnitude methode: ze zijn conceptueel eenvoudiger door de chipfabrikant te realiseren.

2.2.2 Coderen van karakters

De belangrijkste code voor de binaire voorstelling van karakters is *ASCII*.
De *ASCII* code is een 7 bit code en kent dus $2^7 = 128$ verschillende karakters. indien men een ASCII karakter wil voorstellen met 8 bit, wordt de MSB op 0 gezet.
In appendix A van de cursus is ter informatie de volledige *ASCII* code voorgesteld.

De ASCII-code voor een spatie (SP) of blanco is bijvoorbeeld 0100000. We zien in de tabel een kolom met controle karakters, bijvoorbeeld lf (LF) = linefeed, cr (CR) = carriage return, esc (ESC) = escape. Verder is er een kolom met speciale karakters en de cijfers 0 tot 9. Tenslotte vindt men in de laatste 2 kolommen de hoofdletters en kleine letters van het alfabet.

2.3 Oefeningen

2.3.1 Talstelsels

□ Schrijf in binair en hexadecimaal: 124_{10}

□ Schrijf decimaal: 101010_2, $12E_{16}$

□ [Optelling] Reken uit:
$101101_2 + 10111_2$
$3D2F_{16} + 5EA1_{16}$
$1101100_2 + 111111_2$ $11010011_2 + 101101_2$
$CD12_{16} + 2EFF_{16}$ $A2BE_{16} + 1357_{16}$

□ [Aftrekking] Reken uit:
$1110111_2 - 101101_2$
$FC12_{16}+AB5C_{16}$
$1001001_2 - 1111_2$ $1010110_2 - 100111_2$
$12FCD_{16}-FCA4_{16}$ $FC89A_{16}-ACCD5_{16}$

2.3.2 Coderen van gegevens

□ Geef de hexadecimale waarde van -18 met behulp van sign-magnitude, twos-complement en ones-complement.

□ Zet de volgende tekst om naar ASCII: *MPS is leuk!*

Hoofdstuk 3

Coderen van instructies

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- assemblerinstructies kan coderen naar hun hexadecimale waarden.
- hexadecimale waarden kan decoderen naar assemblerinstructies, inclusief de juiste registerbenamingen.

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **3** uren in beslag nemen.

☐ Lees hoofdstuk 1,2 en 3 van de cursus.

☐ Bekijk appendix C van de cursus. Deze lijst geeft de codering aan van de instructies.

☐ Maak de opdrachten over het coderen in paragraaf 3.1

☐ Bekijk appendix B van de cursus. Deze tabel helpt bij het decoderen van hexadecimale instructies. De waarden boven de grijze kolommen geven aan welke bits je moet opzoeken in die kolom. Begin altijd in de linkerkolom en volg de pijlen indien nodig. De parameters van de instructies kan je opzoeken in appendix C.

☐ Maak de opdrachten over het coderen in paragraaf 3.2

3.1 Coderen

□ Geef de hexadecimale codering van de volgende instructies:

- `add $t2, $t1, $t0`
- `add $v0, $a1, $a0`
- `addu $t2, $t1, $t0`
- `andi $s1, $s0, 13`
- `msub $a2, $a3`
- `sra $t1, $t0, 2`
- `lw $t0, 4($a0)`
- `lui $v0, 4097`

□ Gegeven de volgende code:

```
Loop: nop                   # adres Loop [0x0040003c]
      nop                   # opvulling
      bgez  $a0, Loop
      j     Loop
```

- Geef de hexadecimale codering van de bgez instructie
- Geef de hexadecimale codering van de j instructie

3.2 Decoderen

□ Geef de instructies die horen bij volgende hexadecimale codering:

- `0x00a41020`
- `0x2085000a`
- `0x000f8402`
- `0x0461fff7`
- `0x0c100009`
- `0x80480002`
- `0x00000000`

Hoofdstuk 4

SPIM

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- vertrouwd bent met SPIM.

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **4** uren in beslag nemen.

☐ Lees hoofdstuk 4 van de cursus.

4.1 SPIM basisoefeningen

☐ Download en installeer PCSPIM van de volgende URL: http://www.cs.wisc.edu/~larus/spim.html. SPIM is ook beschikbaar op de studenten PC's in de computerlokalen.

☐ Optioneel kan je ook MIPSter downloaden en installeren van de volgende URL: http://www.downcastsystems.com/mipster/. Dit programma is niet strikt noodzakelijk, eender welke teksteditor kan gebruikt worden.

☐ Open SPIM en open voorbeeldprogramma 1 op de site.

☐ Voer het programma uit. De invoer moet je ingeven in de console.

☐ Herinitialiseer SPIM en gebruik F10 om door het programma te lopen. Bekijk hoe de instructies de registers aanpassen en hoe de jump/branch instructies de program counter aanpassen.

☐ Probeer te achterhalen hoe de instructies *jal* en *jr* precies werken.

- ☐ Herinitialiseer SPIM en zet een breakpoint op de regel met commentaar *zet som in $a0*. Het adres kan je opzoeken in het tweede paneel.
- ☐ Voer het programma uit tot SPIM stopt. Kijk in register $t0 (eerste paneel). Deze bevat het resultaat. Bekijk of het resultaat correct is.
- ☐ Laat het programma doorgaan door *continue* te kiezen in het menu *simulator*.
- ☐ Verwijder het breakpoint als het programma afgelopen is.
- ☐ Ga in het derde paneel (het data segment) op zoek naar de tekst *Tot* (ASCII hexadecimaal 0x54 0x6F 0x74).

- ☐ Download de programmatemplate van de site.
- ☐ Plaats de volgende code in het codesegment:

```
      nop
      nop
      nop
      nop
      nop
      li    $a0, 1
Loop: nop
      nop
      bgez  $a0, Loop
      j     Loop
```

- ☐ Open dit programma in SPIM.
- ☐ Bekijk de vertaalde assemblerinstructie van `li $a0, 1` in de derde kolom van het tweede paneel. Li is een pseudoinstructie. Hoe wordt deze instructie geïmplementeerd?
- ☐ Bekijk ook de registerbenamingen in de implementatie van li. Zijn deze benamingen gewijzigd? Waarom gebruiken we deze benamingen niet als we zelf programma's schrijven?
- ☐ Controleer in de tweede kolom van het tweede paneel de codering van de branch en de jump instructies. Vergelijk met je oplossing van vorige week.
- ☐ Verwijder de eerste 5 nop instructies en bekijk de codering van de branch en de jump instructies. Wat is er gewijzigd en wat niet? Hoe komt dit?
- ☐ Plaats tussen `li` en `bgez` een instrunctie die `$a0` met 1 verhoogt. Test je programma.

EXTRA

- □ Je kan de instructies van vorige week ingeven in de template en inladen in SPIM. Controleer je oplossing met de codering zoals gegeven door SPIM.

Hoofdstuk 5

Programmeeropdrachten: Basis

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- enkele basisinstructies kan gebruiken.
- controlestructuren kan schrijven.
- een array kan uitlezen.

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

☐ Lees hoofdstuk 5 van de cursus.

5.1 Controlestructuren

☐ Open de template op de site. We gaan dit programma aanvullen.

☐ Maak een if-then-else-lus met branch instructies:

```
if($t0 > 0)
{
        $t1 = 10
}
else
{
        $t1 = 20
}
$t2 = 30
```

- ☐ Test het programma op verschillende waarden voor $t0. Controleer of $t1 en $t2 de juiste waarden bevatten.

- ☐ Maak een while-lus met branch en jump instructies:

```
$t3 = 0
while($t3 < 30)
{
        $t3 = $t3 + 1
}
```

5.2 Geheugenoperaties

Het is belangrijk een onderscheid te maken tussen registers en geheugen. Beide dienen voor opslag, maar worden anders gebruikt. De meeste instructies nemen enkel registers als parameters, maar de hoeveelheid registers is beperkt. Daarom kan er data in het geheugen worden opgeslagen. Kopiëren van register naar register gebeurt met `move`. Kopiëren van register naar geheugen gebeurt met `sw` en omgekeerd met `lw`.

- ☐ Open voorbeeldprogramma 2 op de site. We gaan dit programma aanvullen.
- ☐ Gebruik de instructie lw om het getal op label *getal1* in $t0 te laden.
- ☐ Gebruik de instructie sw om de waarde in $t0 op te slaan in het geheugen op plaats *getal2*.
- ☐ Voer dit programma uit en bekijk het datasegment. Controleer of de waarde voor *getal1* en *getal2* hetzelfde is.
- ☐ Gebruik de instructie addi om de waarde in $t0 te vermeerderen met 5. Sla op in $t1.
- ☐ Kopieer de waarde in $t0 naar $t2 met de instructie move.
- ☐ Laad 7 in $t1 met de instructie li.
- ☐ Trek $t1 af van $t2 en sla op in $t4.
- ☐ Trek $t2 af van $t1 en sla op in $t5.
- ☐ Bekijk register $t5, en probeer dit getal om te zetten naar decimale waarde.
- ☐ Move is een pseudoinstructie. Gebruik een aantal instructies om $t5 te kopiëren naar $t6 zonder pseudoinstructies te gebruiken.
- ☐ Schrijf een set instructieregels om te testen of de waarde van $t7 deelbaar is door 4. Test het programma met verschillende invoerwaarden voor $t7.

5.3 Arrays

Een array is een blok geheugen dat is ingedeeld in elementen van gelijke grootte. De grootte van de elementen en de lengte van de array is bepaald in je programma en niet in het geheugen zelf. In de komende oefeningen van dit hoofdstuk is de grootte van een element 4 bytes. In een array, en in het geheugen in het algemeen, wordt elke byte apart geadresseerd. Als je een specifiek element in de array wil opvragen, heb je het startadres van dit element nodig. Dit kan berekend worden aan de hand van het startadres van de array en de grootte van de elementen.

- ☐ Open de template op de site. We gaan dit programma aanvullen.
- ☐ Maak een array van 5 getallen in het datasegment en sluit af met een nul.
- ☐ Schrijf een programma dat elk getal uitprint. Stop als je een nul tegenkomt. Gebruik `arraylabel(positie*4)` om een bepaalde plaats in de array uit te lezen. De positie wordt maal 4 gedaan omdat elke plaats in de array 4 bytes inneemt.
- ☐ Breid het programma uit zodat het aantal elementen in de array wordt geteld. Print dit getal uit op het einde.
- ☐ Breid het programma uit zodat het aantal oneven getallen in de array wordt geteld. Print dit getal uit op het einde.

- ☐ Open de template op de site. We gaan dit programma aanvullen.
- ☐ Maak een array met twee 1en, gevolgd door 20 nullen: `.word 1, 1, 0, 0 ... .`
- ☐ Schrijf een programma dat deze array uitprint.
- ☐ Breid het programma uit zodat de array wordt gevuld met de rij van Fibonacci. Gebruik het volgende algoritme:

  1. Lees positie 0 en 1 in van de array in registers.
  2. Bereken de som van deze 2 registers.
  3. Sla op in de array op positie 2.
  4. Schuif 1 plaats op en herhaal. Stop als het einde van de array is bereikt.

Hoofdstuk 6

Subroutines

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- leaf subroutines kunt aanroepen met de juiste conventie.
- leaf subroutines kunt schrijven.

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

☐ Lees hoofdstuk 6 sectie 1, 2, 3 van de cursus.

6.1 Subroutines

☐ In dit hoofdstuk zullen we alleen gebruikmaken van leaf subroutines. Subroutines die subroutines gebruiken komen binnen een paar weken aan bod.

☐ Open voorbeeldprogramma 3 op de site. We gaan dit programma aanvullen.

☐ Vul de main aan zodat de tekst op het label *tekst* wordt omgezet naar hoofdletters. Gebruik de subroutine *toupper*. De subroutine aanroepen moet met de instructie *jal*. Hou er rekening mee dat subroutines de registers $t0 tot $t7 mogen gebruiken, zonder deze op te slaan. De registers $s0 tot $s7 worden wel bewaard over subroutineaanroepen heen; gebruik bij voorkeur deze.

☐ We gaan een extra subroutine toevoegen. Gebruik enkel de registers $t0 tot $t7 in de subroutine, omdat deze niet verondersteld zijn te worden bewaard.

- ☐ Schrijf een functie die een hoofdletter omzet naar een hoofdletter die 2 plaatsen is opgeschoven in het alfabet: A → C, B → D, . . . , X → Z, Y → A, Z → B.
- ☐ Pas deze functie aan zodat het aantal plaatsen dat wordt opgeschoven via een parameter kan worden doorgegeven.
- ☐ Test de subroutine in voorbeeldprogramma 3.

- ☐ Open de template op de site en reserveer geheugen voor een matrix van 5x5 getallen. Vul deze matrix met nullen. De matrix wordt opgeslagen als een opeenvolging van 25 getallen, maar stellen een 2 dimensionale structuur voor. Een specifiek element M(rij, kolom) kan je opvragen met de index in het geheugen: rij + kolom * 5.
- ☐ Schrijf een subroutine die deze matrix uitschrijft naar de console. De matrixstructuur moet zichtbaar zijn. De parameter is het startadres in het geheugen.
- ☐ Schrijf een subroutine die een waarde zet op een specifieke locatie in de matrix. De rij en de kolom moeten als parameter worden doorgegeven, samen met de waarde en het startadres van de matrix.
- ☐ Maak een 2de matrix in het geheugen en schrijf een subroutine die 2 adressen als parameter krijgt en test of deze matrices gelijk zijn. Geef het resultaat (0 of 1) terug in $v0.
- ☐ Schrijf een subroutine die een matrix transponeert. Zie http://en.wikipedia.org/wiki/Transpose

Hoofdstuk 7

Tekst

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- met tekst kunt werken in MIPS

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

7.1 Inleiding

Tekst in SPIM wordt in het geheugen opgeslagen als een opeenvolging van karakters van 1 byte.

7.2 Inlezen en uitschrijven van tekst

- ☐ Open de template op de site en reserveer een blok geheugen van 512 bytes.
- ☐ Lees een string in met system call 8. De argumenten zijn $a0 (het adres van het blok geheugen) en $a1 (de grootte van het blok geheugen). Je kan de tekst ingeven in de console.
- ☐ Schrijf de string uit met system call 4.

7.3 Bewerkingen van tekst

- ☐ Schrijf een subroutine die de lengte van een string teruggeeft. Merk op dat deze niet hetzelfde is als de grootte van het blok geheugen. Een string wordt afgesloten met een 0. Je kan een karakter inlezen met `lb`.
- ☐ Schrijf een subroutine die een string omkeert. Je kan de lengte als parameter doorgeven met behulp van je vorige subroutine.
- ☐ Schrijf een subroutine die test of een string een palindroom is. Geef de oplossing terug.

7.4 Lezen en schrijven van karakters

- ☐ Schrijf een subroutine die een bepaald karakter vervangt door een ander karakter. Vraag aan de gebruiker welke positie en het nieuw karakter. Je kan een karakter inlezen met system call 12. Het karakter wordt opgeslagen in $v0.
- ☐ Schrijf een subroutine die elke oneven karakter uitprint. Gebruik system call 11. Het karakter dat je wil uitprinten moet je plaatsen in $a0.

Hoofdstuk 8

Datastructuren

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- dynamisch geheugen kan alloceren.
- datastructuren kan opbouwen en gebruiken.

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

☐ Herhaal, indien nodig, het lesmateriaal over pointers en dynamisch gelinkte lijsten van de cursus *Imperatief Programmeren in C*.

8.1 Statische lijsten

☐ Open de template op de site. We gaan dit programma aanvullen.

☐ Maak in het datasegment een vrije ruimte van 320 bytes. Hier gaan we 5 datarijen van personen in opslaan van elk 64 bytes. Elke rij is ingedeeld als volgt: 32 bytes voor de naam, 4 bytes voor de leeftijd en 28 bytes voor de stad. Geef de data het label *rijen*. Het aantal rijen dat in gebruik is wordt opgeslagen in $s0.

☐ Schrijf in het maingedeelte een menugebaseerde interface. Het programma print de volgende tekst uit:

```
[1] Nieuwe rij toevoegen
[2] Rij uitprinten
```

```
[3] Rij wijzigen
[0] Programma stoppen
```

Vervolgens wordt een getal ingelezen die de actie aangeeft. Elke actie is een subroutine (zie verder). Na de actie moet het menu terugkomen en moet een nieuwe actie mogelijk zijn.

- ☐ Schrijf een subroutine die het programma stopt. Koppel deze subroutine aan de juiste invoer van de gebruiker.
- ☐ Schrijf een subroutine die een nieuwe rij toevoegt. De rij wordt opgeslagen op positie *rijen + 64 * $s0*. De gebruiker moet elk veld ingeven. Veld 1 (naam) heeft adres *rijen + 64 * $s0*; veld 2 (leeftijd) heeft adres *rijen + 64 * $s0 + 32*; veld 3 (stad) heeft adres *rijen + 64 * $s0 + 36*. Controle op de lengte van de strings is niet nodig. Er moet een foutmelding gegeven worden als het blok geheugen vol is. Vergeet niet $s0 te verhogen. Koppel deze subroutine aan de juiste invoer van de gebruiker.
- ☐ schrijf een subroutine die een rij uitprint. De gebruiker geeft een rijnummer in (0-4). Er moet een foutmelding gegeven worden als de rij niet bestaat. De rij begint op positie *rijen + 64 * invoer*. Koppel deze subroutine aan de juiste invoer van de gebruiker.

EXTRA

- ☐ Schrijf een subroutine die een rij kan wijzigen. De gebruiker geeft een rijnummer in (0-4), en het programma vraagt de waarde voor de 3 velden opnieuw in te geven. Als de rij niet bestaat, moet een foutmelding worden weergegeven. Koppel deze subroutine aan de juiste invoer van de gebruiker.

8.2 Dynamisch gelinkte lijsten

- ☐ Open de template op de site. We gaan dit programma aanvullen.
- ☐ In dit programma worden de rijen niet in het statische datasegment opgeslagen, maar in het dynamische gedeelte. Je kan een stuk geheugen aanvragen met de system call *sbrk* (code 9). Sla in $a0 de grootte op van het blok dat je wil reserveren en je krijgt het adres ernaar terug in $v0. De grootte is altijd 68 bytes en is als volgt ingedeeld: 32 bytes voor de naam, 4 bytes voor de leeftijd, 28 bytes voor de stad en 4 bytes om de rijen aan elkaar te koppelen (dit veld noemen we de next-pointer).
- ☐ We gaan het register $s0 gebruiken om het adres van de eerste rij op te slaan. Als er geen rijen zijn, is dit register 0.
- ☐ Schrijf in het maingedeelte een menugebaseerde interface. Het programma print de volgende tekst uit:

```
[1] Nieuwe rij toevoegen
[2] Rijen uitprinten
[3] Leeftijd zoeken
[0] Programma stoppen
```

Vervolgens wordt een getal ingelezen die de actie aangeeft. Elke actie is een subroutine (zie verder). Na de actie moet het menu terugkomen en moet een nieuwe actie mogelijk zijn.

- □ Schrijf een subroutine die het programma stopt. Koppel deze subroutine aan de juiste invoer van de gebruiker.

- □ Schrijf een subroutine die een nieuwe rij toevoegt:

  1. Vraag een nieuw blok geheugen aan van 68 bytes. Dit adres bevindt zich in $v0.
  2. Sla in $v0 + 64 (de next-pointer) de waarde op van $s0.
  3. Sla in $s0 de waarde op van $v0. $s0 bevat nu het adres van de nieuw toegevoegde rij. Het adres van de voorlaatst toegevoegde rij is opgeslagen in de laatst toegevoegde rij. Als je bijvoorbeeld 3 rijen hebt toegevoegd, zal $s0 het adres van rij 3 bevatten, rij 3 het adres van rij 2 bevatten, rij 2 het adres van rij 1, en rij 1 het adres 0 opslaan om aan te geven dat dit de eerste rij is.
  4. Sla de andere gegevens op de juiste plaats op. De rij start op het adres zoals is opgeslagen in $s0.

- □ Schrijf een subroutine die elke rij uitprint. Gebruik register $t0 om de huidige rij aan te geven. Het adres van de eerste rij staat in $s0. Kopieer dit adres naar $t0. Print de rij op dit adres uit en plaats het adres in de next-pointer in $t0. Blijf dit herhalen tot $t0 0 is.

- □ Schrijf een subroutine die een leeftijd zoekt. Loop op dezelfde manier door de lijst als in de vorige subroutine, maar print enkel de rij als de leeftijd overeenkomt met de ingegeven waarde.

Hoofdstuk 9

Subroutines, deel 2

Doelstellingen

Na het bestuderen van dit hoofdstuk wordt verwacht dat je:

- subroutines kunt aanroepen en schrijven in subroutines met behulp van de stack.
- simpele recursie kan gebruiken.

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

☐ Lees hoofdstuk 6 van de cursus. Er zullen geen oefeningen zijn over sectie 6.7 (Re-entrant functies).

9.1 Subroutines

☐ In dit hoofdstuk zullen we gebruikmaken van subroutines in subroutines. Vanaf nu **moet** aan het begin van de subroutine de inhoud van $ra bewaard worden op de stack en terug hersteld worden op het einde. Bewaar ook alle registers die je gebruikt. t-registers moet je bewaren als je een subroutine aanroept, en s-registers moet je bewaren in de subroutine zelf (indien nodig). Bewaren gebeurt door deze op de stack te zetten. Bekijk hiervoor de voorbeeldcode uit de cursus.

☐ Open de template op de site. We gaan dit programma aanvullen.

☐ Schrijf een subroutine die twee karakters controleert op gelijkheid, zonder te letten op hoofdletters (a = A). Geef 1 terug als ze gelijk zijn, anders 0. Je kan elke letter omzetten naar lowercase en testen op gelijkheid.

□ Schrijf met behulp van de vorige subroutine een nieuwe subroutine die een karakter zoekt in een gegeven string, zonder te letten op hoofdletters. Geef de positie terug. De parameters zijn het te zoeken teken, het adres naar de string en de lengte van de string. Geef -1 terug als niet gevonden of als de lengte nul is.

```
int Zoek(teken, adres, lengte)
{
   if(lengte <= 0) return -1;

   for(i = 0; i < lengte; i++)
   {
      if(Vergelijk(teken, adres[i]) == 1)
         return i;
   }

   return -1;
}
```

□ Schrijf met behulp van de vorige subroutine een nieuwe subroutine die het aantal voorkomens van een teken in een string telt, zonder te letten op hoofdletters.

```
int Tel(teken, adres, lengte)
{
   if(lengte <= 0) return 0;
   teller = 0;

   while(lengte > 0)
   {
      ret = Zoek(teken, adres, lengte);
      if(ret == -1)
         break;
      adres += ret + 1;
      lengte -= ret + 1;
      teller++;
   }
   return teller;
}
```

□ Open de template op de site. We gaan dit programma aanvullen.

□ Schrijf een programma dat controleert of de haakjes in een gegeven string bij elkaar passen. Gebruik alleen ronde haakjes. Los het probleem op met behulp van een teller.

Opmerking: Hou er rekening mee dat SPIM karakters niet sequentieel opslaat.

```
Invoer: dit ( is een ) test
uitvoer: ja

invoer: dit ( is een test
uitvoer: nee

invoer: )(
uitvoer: nee

invoer: dit ( is ( een ) ()test)
uitvoer: ja
```

□ Breid het programma uit, zodat ook rechte en gekrulde haakjes gebruikt worden. Gebruik de stack om dit probleem op te lossen. Vergeet niet het einde van de stack aan te geven.

9.2 Recursie

□ Schrijf een subroutine die een element uit de Fibonaccirij teruggeeft, met behulp van het volgende recursieve algoritme:

```
int Fibo(int position)
{
   if(position < 2)
      return 1;
   else
      return Fibo(position - 1) + Fibo(position - 2);
}
```

Vergeet niet elk register dat je gebruikt te bewaren op de stack en terug te herstellen na een subroutineaanroep.

Hoofdstuk 10

Combinatieoefeningen

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

- ☐ Deze oefeningen combineren alle concepten uit de volledige cursus.
- ☐ Schrijf een programma dat een machtsberekening uitvoert. Het programma moet 2 getallen inlezen en 1 getal teruggeven. Gebruik system calls voor het inlezen en uitschrijven; gebruik een lus om de machtsberekening uit te voeren.

- ☐ Een *linear feedback shift register* is een register waarvan de inhoud een functie is van zijn vorige inhoud. Dit register kan gebruikt worden om pseudowillekeurige getallen te berekenen, als de functie juist gekozen wordt. We kiezen in dit geval voor de volgende bewerking van het register:

```
unsigned int shiftregister = 1;
unsigned int bit;
do
{
   bit  = ((shiftregister >> 0) ^ (shiftregister >> 10)
      ^ (shiftregister >> 30) ^ (shiftregister >> 31) ) & 1;
   /* ^ = XOR, >> = shift right */
   shiftregister =  (shiftregister >> 1) | (bit << 31);
   printf("%d\n", shiftregister);
} while(true);
```

Hier wordt een nieuwe bit berekend met behulp van een aantal bits uit het register. Dit bit wordt vooraan het register ingeschoven en creeert zo een nieuwe waarde in het register. De initiele waarde is 1. Na elke stap in het algoritme bevat het register waarden die bruikbaar zijn als pseudowillekeurige getallen.

1. Implementeer dit algoritme, zodat de eerste 10 waarden worden uitgeprint, zijnde:

   ```
   -2147483648
   -1073741824
   1610612736
   -1342177280
   -671088640
   1811939328
   -1241513984
   -620756992
   1837105152
   -1228931072
   ```

2. Wat is de eerste waarde die kleiner is dan 100000 en groter dan 0?

□ Maak een array (label *arr*) van 20 getallen van 4 bytes. Schrijf een programma dat deze array sorteert met behulp van het volgende algoritme: Vergelijk de eerste 2 getallen van de array. Als het eerste groter is dan het tweede, wissel deze getallen om. Doe dit voor elk paar opeenvolgende getallen in de array. Blijf herhalen tot er geen wissels meer zijn uitgevoerd. Dit algoritme noemt bubble sort. Voorbeeld:

[3 2] 6 5 4	$3 > 2$, wissel om
2 [3 6] 5 4	ok
2 3 [6 5] 4	$6 > 5$, wissel om
2 3 5 [6 4]	$6 > 4$, wissel om, en herhaal
[2 3] 5 4 6	ok
2 [3 5] 4 6	ok
2 3 [5 4] 6	$5 > 4$, wissel om
2 3 4 [5 6]	ok, en herhaal
[2 3] 4 5 6	ok
2 [3 4] 5 6	ok
2 3 [4 5] 6	ok
2 3 4 [5 6]	ok, en geen wissels, dus stop

Hoofdstuk 11

Combinatieoefeningen 2

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

□ Deze oefeningen combineren alle concepten uit de volledige cursus.

11.1 Het vermoeden van Collatz

Kies een positief getal n.

$$f(n) = \begin{cases} n/2 & \text{als n even} \\ 3n+1 & \text{als n oneven} \end{cases}$$

Het vermoeden van Collatz zegt dat je uiteindelijk op 1 uitkomt als je deze operatie recursief blijft herhalen.

Bijvoorbeeld:

$$106 \to 53 \to 160 \to 80 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

Implementeer deze bewerking op een recursieve manier, en stop als je 1 bereikt hebt. Print elk getal uit.

11.2 Torens van Hanoi

Als je niet weet wat de torens van Hanoi zijn, bekijk dan http://nl.wikipedia.org/wiki/Torens_van_Hanoi. We gaan 2 programma' schrijven die de oplossing uitprinten: een iteratief en een recursief programma. De oplossing wordt uitgeprint als een lijst van zetten (schijf: van toren → naar toren). Schijf 1 is de kleinste. Start altijd met alle schijven in toren 1 en eindig met alle schijven in toren 3. Zorg dat

je programma's met elke hoeveelheid schijven overweg kan. Voor 3 schijven is de uitvoer:
1: 1 → 3
2: 1 → 2
1: 3 → 2
3: 1 → 3
1: 2 → 1
2: 2 → 3
1: 1 → 3

11.2.1 Iteratief

Het iteratieve algoritme is als volgt:

1. Verplaats schijf 1 naar links (ga rond als schijf 1 uiters links ligt).
2. Doe de enige mogelijke zet zonder schijf 1 te verplaatsen.

Herhaal tot de oplossing bereikt is. Maak een array met de positie van elke schijf. Gebruik een subroutine om de bovenste schijf per toren te berekenen.

11.2.2 Recursief

Het recursieve algoritme is als volgt:

1. Verplaats de toren op 1 naar 2, behalve de grootste schijf
2. Verplaats de grootste schijf naar 3
3. Verplaats de toren die je in stap 1 hebt verplaatst naar 3

Het verplaatsen van de toren kan recursief worden opgelost. Gebruik hiervoor een subroutine met 3 parameters: torenVan, torenNaar, en hoogteToren. Als hoogteToren 1 is, is de verplaatsing triviaal. Voor dit algoritme is het niet nodig een array van schijven bij te houden; de oplossing kan rechtstreeks worden uitgeprint.

Hoofdstuk 12

Combinatieoefeningen 3

Begrote studietijd

Het bestuderen van het cursusmateriaal en het uitvoeren van de taken zal voor dit hoofdstuk ongeveer **5** uren in beslag nemen.

- ☐ Deze oefeningen combineren alle concepten uit de volledige cursus.

- ☐ Probeer te achterhalen wat dit programma doet (zie ook het programma mystery.s op de site):

```
   .data
arr:
   .word -1, 4, 5, -9 ,2 ,-9 ,-8 ,2, 1, -10, 0, 1, -2, -4, 2
prompt:
   .asciiz "De oplossing is: "

   .text
   .globl main
main:
   lw    $t1, arr
   li    $t2, 0
l1:
   bgez  $t1, l2
   addi  $t0, $t0, 1
l2:
   beqz  $t1, done
   addi  $t2, $t2, 4
   lw    $t1, arr($t2)
   b     l1

done:
```

```
li    $v0, 4
la    $a0, prompt
syscall               # Print de prompt

li    $v0, 1
move  $a0, $t0
syscall               # Print $t0

li    $v0, 10
syscall               # Sluit het programma af
```

□ Schrijf een algoritme dat de waarden van 2 registers omwisselt, zonder gebruik te maken van een tijdelijk register. Dit is mogelijk met behulp van de xor instructie. Gebruik het internet als je niet weet hoe dit moet.

□ Dit is ook mogelijk met add en sub. Schrijf ook hier een programma voor. Tip: gebruik de unsigned versies.

□ Zet de volgende subroutine om naar assemblerinstructies. Geef alle parameters door via registers. A is een array van getallen van 4 bytes. Schrijf ook een programma dat deze subroutine test.

```
int Min(int* A, int low, int high)
{
  if (low == high)
    return A[low];
  int mid = (low+high)/2;
  int min1 = Min( A, low, mid);
  int min2 = Min( A, mid + 1, high);
  if (min1 > min2)
    return min2;
  return min1;
}
```

De functieaanroep gebeurt als volgt:

```
arraylabel:
  .word 1 2 3 4 5 1 2 3 4 5
.....
la  $a0, arraylabel
li  $a1, 0
li  $a2, 9
jal min
```

```
# minimum van array in $v0

min:
.....
```