# 

**Faculteit FNT ~~TCTIa7v1~~**

|  |  |
| --- | --- |
| Collegejaar | 2017-2018 |
| Periode | A |
| Opleiding | ICT |
| Osiris Cursusnaam | C++ Programmeren & SE 1 |
| Osiris Cursuscode | TCTI-V2CPSE1-16 |
| Tijdsduur tentamen | 120 minuten |
| Tentamendatum | ~~18-10-2017~~ |
| Tijdstip | ~~Van 13:00-15:00~~ |
|  |  |
| **Invullen door de docent** |  |
| Meerdere versies | nee |
| Docent(en) | Wouter van Ooijen |
| Aantal pagina’s (incl. voorblad) | 14  De laatste 4 mogen worden losgemaakt en hoeven niet ingeleverd te worden. |
| Rekenmachine | nee |
| Andere toegestane hulpmiddelen | nee |
| Aantal vragen | 2 open vragen, 20 vier-keuze vragen |
| Soort vragen | open vragen: Uitwerkpapier (lijntjes)  vier-keuze vragen: antwoordvel ~~(blz 10)~~ |
| Antwoorden uitwerken op | ja |
| Kladpapier toegestaan | nee |
|  |  |
|  |  |

**Invullen door de student**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Studentnummer en naam student |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Naam docent |  | | | | | | | |

|  |  |
| --- | --- |
|  | **Tentamenopgave mag NIET worden meegenomen, gefotografeerd, overgeschreven of op andere wijze gedupliceerd worden!** |

|  |
| --- |
| Bij iedere open vraag is aangegeven hoeveel punten je met die vraag kunt behalen. Het totaal voor de open vragen is 45.  Als je antwoord code bevat zal dit niet beoordeeld worden op ‘kleine’ syntaxfouten, d.w.z. die fouten die door de compiler gesignaleerd zouden worden, en waarvan de verbetering vanzelf spreekt.  Bij open vragen waar ja/nee, een getal, of iets dergelijks als antwoord wordt gevraagd is het aan te bevelen ook aan te geven hoe je aan dit antwoord bent gekomen.  Mogelijk zijn delen van de vraag **vet** gedrukt of gemarkeerd met \*sterretjes\*. Het is aan te raden daar extra op te letten.  De 20 multiple-choice vragen kunnen maximaal 45 punten opleveren. Een goed antwoord verhoogt je score, een fout (of geen) antwoord verlaagt je score niet. Vul dus altijd een antwoord in. De antwoorden op de multiple-choice vragen vult je in in de tabel achteraan. Als je een multiple-choice antwoord wilt verbeteren kruis je het foute antwoord door en vult je daarachter het goede antwoord in.  Als een vraag (of bij multiple-choice ook: een of meer antwoorden) volgens jou een fout bevat die je hindert bij het geven van uw antwoord schrijf dit dan op.  Het eindcijfer is het totaal aantal punten plus 10, gedeeld door 10. Het minimum is dus 1.0. |

1. [25] Schrijf een class template voor een *verzameling* klasse Abstract Data Type. Zo’n klasse kan waarden van een bepaald type opslaan, tot een bepaald maximum aantal waarden. Het type en het maximum zijn template parameters.

Schrijf een default constructor, die de verzameling leeg maakt.

Schrijf een contains(..) functie die teruggeeft of het meegegeven element in de verzameling voorkomt.

Schrijf een add(…) functie die het meegegeven element aan de verzameling toevoegt. Als de verzameling al het maximum aantal elementen bevat dan heeft deze functie geen effect. Let op dat een verzameling een bepaald element maximaal 1 keer kan bevatten.

Schrijf een remove(..) functie die het meegegeven element uit de verzameling verwijdert. Als het element niet in de verzameling voorkomt dan heeft deze functie geen effect.

Voor zover mogelijk moeten de functies bruikbaar zijn op een const object.

Tips:

* Hou de opgeborgen waarden bij in een array en hou bij hoeveel elementen van dat array geldig zijn.
* Schrijf eerst de constructor en contains()
* add() is simpel maar let op dat je geen element toevoegt dat er al in zit
* een manier om remove(…) te schrijven is met twee indexen (een voor lezen en een voor schrijven) door het array heen te lopen.

1. [20] Het inner-product van twee (even lange) vectoren is de som van de paarsgewijze producten van de elementen. In C:

|  |
| --- |
| float inner\_product( float a[], float b[], int length ){  float sum = 0;  for( int i = 0; i < length ){  sum += a[ i ] \* b[ i ];  }  return sum;  } |

Schrijf een inner\_product subroutine in Cortex M0 assembler. De chip beschikt niet over floating point instructies, maar je mag gebruik maken van een float constante 0.0 en subroutines voor float vermenigvuldigen en float optellen. Deze subroutines volgen (natuurlijk) de calling convention. Een float is 32 bits.

|  |
| --- |
| // floating point 0.0  float\_zero: .word . . .  // float float\_multiply( float a, float b ){ return a \* b; }  float\_multiply: . . .  // float float\_add( float a, float b ){ return a + b; }  float\_add: . . .  .cpu cortex-m0  .text  .align 1  .global inner\_product:  inner\_product:  (hier komt jouw code) |

Voer je antwoorden in op de laatste bladzijde, dat is de lijst die wordt nagekeken.

|  |
| --- |
| 1) Wat is waar over de preprocessor? |
| A: Die berekent expressies die uitsluitend constanten bevatten. B: Die wordt door de linker (automatisch) aangeroepen. C: Die wordt door de compiler (automatisch) aangeroepen. D: Die voorkomt (automatisch) dat header files meerdere keren worden ge-include. |

|  |
| --- |
| 2) Wat is het gevolg als een variabele in een applicatie wel gedefinieerd maar nergens gedeclareerd is? |
| A: Een foutmelding van de compiler. B: Niets, dit is legaal. C: Een foutmelding van de linker. D: Een run-time error. |

|  |
| --- |
| 3) Welke uitspraak is waar over globale objecten? |
| A: Zulke objecten hebben nooit een adres in het geheugen omdat ze in CPU registers worden opgeslagen. B: De compiler bepaalt het adres in het geheugen van zulke objecten. C: Het adres in het geheugen van zulke objecten wordt bepaald door de waarde in de SP (Stack Pointer). D: De linker bepaalt het adres in het geheugen van zulke objecten. |

|  |
| --- |
| 4) Welke C functie komt NIET overeen met deze assembler subroutine? |
| ADD R0, R0, R1  MOV PC, LR |
| A: char f( char a[], int b ){ return a[ b ]; } B: unsigend char f( unsigned char a, unsigned char b ){ return a + b; } C: char \* f( char a[], int b ){ return & a[ b ]; } D: int f( int a, int b ){ return a + b; } |

|  |
| --- |
| 5) Wat is een taak van de make tool en de bijbehorende Makefile? |
| A: Controleren of de juiste files genoemd worden in #include regels. B: Alle files die niet up-to-date zijn (opnieuw) aanmaken. C: Alle files compileren en linken. D: Controleren of headers files de juiste #ifndef guards bevatten. |

|  |
| --- |
| 6) Wat is waar met betrekking tot de volgende code?  (De . . . geven aan dat daar wat is weggelaten.) |
| template< int N > void some\_function( . . . ){ . . . } |
| A: Dit declareert een reeks functies. B: Dit definieert een reeks functies. C: Dit declareert 1 functie. D: Dit definieert 1 functie. |

|  |
| --- |
| 7) Wat is waar met betrekking tot templates? |
| A: Een functie template kan je alleen aanroepen als je de template parameters aangeeft. B: Een klasse template kan je instantieren (een variabele aanmaken van dat type) zonder \*alle\* template parameters aan te geven. C: Een functie template kan je aanroepen zonder template parameters aan te geven. D: Een klasse template kan je instantieren (een variabele aanmaken van dat type) zonder template parameters aan te geven. |

|  |
| --- |
| 8) Wat is het doel van het proxy pattern? |
| A: Toegang te bieden tot een object of functionaliteit via een ander object. B: Het gebruik van een object door verschillende taken synchronizeren. C: Het vermijden van run-time berekeningen (door ze te vervangen door compile-time berekeningen). D: Bepalen wat de afstand tussen twee objecten is. |

|  |
| --- |
| 9) In een preemptief multitasking systeem worden interrupts gebruikt om te reageren op externe events. Welk mechanisme wordt hiervoor gebruikt in een cooperatief systeem? |
| A: stack switching B: event flags C: polling D: channels |

|  |
| --- |
| 10) Bij een Cortex-M0 zal een context switch in een preemptief multitasking systeem iets meer tijd kosten dan op een cooperatief systeem. Waarom? |
| A: Er moeten meer registers worden bewaard. B: Er is meer gevaar voor deadlock. C: De bewering is niet waar: een context switch kost in beide gevallen evenveel tijd. D: Er moet worden gepollt. |

|  |
| --- |
| 11) Wat is waar over een functie of methode die als constexpr is gedeclareerd? |
| A: Zijn return waarde mag tijdens het runnen niet worden gewijzigd. B: Hij zal tijdens compilatie worden uitgevoerd. C: Hij zal nooit tijdens compilatie worden uitgevoerd. D: Hij kan tijdens compilatie worden uitgevoerd. |

|  |
| --- |
| 12) Welke fabrikanten maken chips met een Cortex CPU er in? |
| A: Alleen de firma ARM. B: Alleen licentienemers van de firma ARM. C: Potentieel iedereen, want de Cortex is een 'vrij' ontwerp. D: Licentienemers van de firma ARM en ook de firma ARM zelf. |

|  |
| --- |
| 13) Welk nut kan een instructie 'ADD R0, R0, #0' hebben? |
| A: De pipeline wordt geflusht. B: De waarde in R0 wordt 'sign extended'. C: De flags worden gezet. D: Er wordt gecheckt of R0 wel een geldig geheugen adres bevat. |

|  |
| --- |
| 14) Het geheugen adres dat wordt gebruikt in een word load of store instructie moet |
| A: Deelbaar zijn door 4. B: Deelbaar zijn door 2. C: Deelbaar zijn door 8. D: Deelbaar zijn door 1 (dat zijn dus alle adressen). |

|  |
| --- |
| 15) Wanneer is het nodig om, aan het begin van een subroutine, het LR register te bewaren op de stack? |
| A: Als de subroutine een andere subroutine aanroept. B: Als de subroutine door een andere subroutine wordt aangeroepen. C: Als de subroutine een B (branch) instructie bevat. D: Als de subroutine registers op de stack bewaart. |

|  |
| --- |
| 16) Waar worden, bij het linken van een microcontroller applicatie, de adressen van BSS (0-initialized data) segment objecten geplaats? |
| A: Niet, dat gebeurt pas bij het runnen. B: Zowel in ROM als in RAM. C: In RAM. D: In ROM (Flash). |

|  |
| --- |
| 17) Wat is de default voor een assembler label? |
| A: Globaal (zichtbaar voor alle files). B: Alleen lokaal zichtbaar (in de file zelf). C: Default lokaal, tenzij het in een 'extern "C"' declaratie wordt vermeldt. D: Een label voor een B is lokaal, een label voor een BL is globaal. |

|  |
| --- |
| 18) Wat is het effect als in een subroutine header de regel .text wordt vervangen door .data ? |
| A: De applicatie crasht als de subroutine wordt aangeroepen, want je kan geen code uitvoeren vanuit het .data segment. B: De code van de subroutine wordt in RAM geplaats. C: De ruimte voor de code wordt nu (waarschijnlijk) door andere .data segment declaraties (her-) gebruikt, waardoor de code overschreven zal worden. D: Dat veroorzaakt een foutmelding tijdens het assembleren (= compileren van een assembler programma). |

|  |
| --- |
| 19) Welke instructie is een logische vertaling van het lezen van een element uit een integer array (de index is een berekende waarde)? |
| A: ADD R1, R2, #16 B: ADD R1, R2, R3 C: LDR R1, [ R0, #16 ] D: LDR R1, [ R0, R2 ] |

|  |
| --- |
| 20) Hoeveel kloks kost het uitvoeren van de volgende code op een Cortex-M0 (3 stage pipeline, enkelvoudig toegangkelijk geheugen)? |
| LDRB R0, [ R1 ]  STR R0, [ R2 ]  ADD R1, R1, #1  ADD R2, R2, #4 |
| A: 6 B: 8 C: 4 D: 9 |

|  |  |
| --- | --- |
| **Muliple Choice Vraag** | **Antwoord (schrijf duidelijk: A, B, C, of D)** |
| 1. |  |
| 2. |  |
| 3. |  |
| 4. |  |
| 5. |  |
| 6. |  |
| 7. |  |
| 8. |  |
| 9. |  |
| 10. |  |
| 11. |  |
| 12. |  |
| 13. |  |
| 14. |  |
| 15. |  |
| 16. |  |
| 17. |  |
| 18. |  |
| 19. |  |
| 20. |  |

**That’s all folks,  /  ?**

De volgende bladzijden mogen (om makkelijker te bladeren) door de student losgehaald worden en hoeven niet ingeleverd te worden.

**Cortex M0 assembly summary**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Low  registers | |  |  |  |  | |
|  |  | R0 |  | Arguments,  Scratch  (not preserved) | |
|  | R1 |
|  | R2 |
|  | R3 |
|  | R4 |  | Local variables (preserved) | |
|  | R5 |
|  | R6 |
|  | R7 |
|  |  | |  | R8 |
|  |  | |  | R9 |
|  |  | |  | R10 |
|  |  | |  | R11 |
|  |  | |  | R12 |  | reserved | |
|  |  | |  | R13 |  | Stack Pointer | |
|  |  | |  | R14 |  | Link Register | |
|  |  | |  | R15 |  | Program Counter | |
|  |  | |  |  |  |  | |
| **Segment / Region** | | **Used …** | | | | | **Present in executable** | | **Allocated at** | **Allowed operations** |
| CODE | | To store executable code | | | | | Yes | | Build | Execute |
| RODATA | | To store read-only data | | | | | Yes | | Build | Read |
| DATA | | To store data that has an initial value, but can be changed | | | | | Yes | | Build | Read, write |
| BSS | | To store data that has no specific initial value | | | | | No | | Build | Read, write |
| STACK | | For or the stack (local data and return addresses) | | | | | No | | Run | Read, write |
| HEAP | | For the heap (dynamically allocated data) | | | | | No | | Run | Read, Write |

|  |  |
| --- | --- |
| **Directive** | **Effect** |
| .align N | Skip addresses, until the address is divisible by N. |
| .ascii ”text”, ”text2”, … | Produce the bytes that contain the ASCII characters in each string. |
| .asciz ”text”, ”text2”, … | As .ascii, but add a ’\0’ byte after each string. |
| .bss | Following items are produced in the BSS segment (read/write, 0-initialized). |
| .byte N, M, … | Produce a number of bytes, each with one of the specified values. |
| .cpu cortex-m0 | Specify that the target CPU is a Cortex-M0. |
| .data | Following items are produced in the DATA segment (read/write, initialized). |
| .global Name, Name2, … | Make the indicated local labels accessible to other source files. |
| .hword N, M, … | Produce a number of half-words, each with one of the specified values. |
| .word N, M, … | Produce a number of words, each with one of the specified values. |
| .text | Following items are produced in the CODE segment (read-only). |

|  |  |
| --- | --- |
| **Data movement** | |
| MOV Rd, Rm | Copy the content of Rm to Rd |
| MOV Rd, #immed8 | Load the value immed8 into Rd |
| LDR1 Rd, [ Rn, Rm ] | Load Rd with the content of the memory address Rn + Rm |
| LDR1 Rd, [ Rn, #immed52 ] | Load Rd with the content of memory address Rn + immed52 |
| LDR Rd, [ PC3, #immed8 ] | Load Rd with the content of memory address PC + 4 + immed8 \* 4 |
| STR1 Rd, [ Rn, Rm ] | Store the content of Rd in memory at address Rn + Rm |
| STR1 Rd, [ Rn, #immed52 ] | Store the content of Rd in memory at address Rn + immed52 |
| STR Rd, [ PC3, #immed8 ] | Store the content of Rd in memory at address PC + 4 + immed8 \* 4 |
| LDR Rd, =immed32 | Load Rd with the value immed32 |
| 1Append H for a half-word (16-bit) access, B for a byte (8-bit) access.  2The immed5 value in the opcode is multiplied by 4 for a word access, by 2 for a half-word access.  3Can also be SP, which is useful for compiled languages for accessing local variables the stack. | |
| LDRSH Rd, [ Rn, Rm ] | Load Rd with the sign-extended half-word content of the memory address Rn + Rm |
| LDRSB Rd, [ Rn, Rm ] | Load Rd with the sign-extended byte content of the memory address Rn + Rm |
| PUSH { low\_register\_list1 } | Push the indicated registers onto the stack |
| PUSH { low\_register\_list1, LR } | Push the indicated registers and the LR onto the stack |
| POP { low\_register\_list1 } | Pop the indicated registers from the stack |
| POP { low\_register\_list1, PC } | Pop the indicated registers and the PC from the stack |
| 1List of low registers, separated by commas, for instance ”R2, R3, R4, R6”. A range can be specified, for instance ”R2 – R4, R6”. | |
| LDM Rn, { low\_register\_list } | Load the indicated registers from the memory pointer to by Rn and subsequent memory locations. |
| LDMIA Rn!, { low\_register\_list } | Load the indicated registers from the memory pointer to by Rn and subsequent memory location, then update Rn to point to the next higher memory address. |
| STMIA Rn!, { low\_register\_list } | Save the indicated registers from the memory pointer to by Rn and subsequent memory location, then update Rn to point to the next higher memory address. |

|  |  |
| --- | --- |
| **Arithmetic** | |
| ADD1 Rd, Rn, Rm | Rd = Rn + Rm |
| ADD Rd, Rn, #immed3 | Rd = Rn + immed3 |
| ADD Rd, Rd, #immed8 | Rd = Rd + immed8 |
| ADC Rd, Rn, Rm | Rd = Rn + Rm + Carry |
| SUB Rd, Rn, Rm | Rd = Rn - Rm |
| SUB Rd, Rn, #immed3 | Rd = Rn - immed3 |
| SUB Rd, Rd, #immed8 | Rd = Rd - immed8 |
| SBC Rd, Rn, Rm | Rd = Rn - Rm - Carry |
| MUL Rd, Rn, Rm | Rd = Rn \* Rm 2 |
| 1 For ADD only, all registers are allowed  2 The lower 32 bits of the result are kept, the higher 32 bits are ignored | |

|  |  |
| --- | --- |
| **Logical** | |
| AND Rd, Rn, Rm | Rd = Rn & Rm |
| EOR Rd, Rn, Rm | Rd = Rn | Rm |
| ORR Rd, Rn, Rm | Rd = Rn ^ Rm |
| BIC Rd, Rn, Rm | Rd = Rn & ! Rm |
| MOVN Rd, Rn | Rd = ! Rn |
| All logical operations are bitwise: bit N in the results is affected only by bits N in the source(ses). | |

|  |  |
| --- | --- |
| **Shift** | |
| LSL Rd, Rn, Rm | Rd = Rn << Rm |
| LSL Rd, Rn, #immed5 | Rd = Rn << immed5 |
| LSR Rd, Rn, Rm | Rd = Rn >> Rm |
| LSR Rd, Rn, #immed5 | Rd = Rn >> immed5 |
| ASR Rd, Rn, Rm | Rd = Rn / 2 \*\* Rm |
| ASR Rd, Rn, #immed5 | Rd = Rn / 2 \*\* immed5 |
| ROR Rd, Rn, Rm | Rd = ( Rn >> ( Rm % 32 )) | ( Rn << ( 32 – ( Rm % 32 ))) |

|  |  |
| --- | --- |
| **Extend** | |
| SXTH Rd, Rm | Sign-extend the half-word 2’s-complement value in Rm , store in Rd |
| SXTB Rd, Rm | Sign-extend the byte 2’s-complement value in Rm , store in Rd |
| UXTH Rd, Rm | Zero-extend the half-word unsigned value in Rm , store in Rd |
| UXTB Rd, Rm | Zero-extend the byte unsigned value in Rm , store in Rd |

|  |  |
| --- | --- |
| **Compare and test** | |
| TST Rn, Rm | Calculate Rn & Rm |
| CMP Rn, Rm | Calculate Rn - Rm |
| CMN Rn, Rm | Calculate Rn + Rm |
| CMP Rn, #immed8 | Calculate Rn – immed81 |
| 1range -128 … +127 | |

|  |  |
| --- | --- |
| **Add and subtract SP** | |
| ADD SP, SP, #immed71 | SP = SP + immed7 |
| SUB SP, SP, #immed71 | SP = SP - immed7 |
| ADD Rd, SP, #immed82 | Rd = SP + immed8 |
| 1range 0 ... 512 in steps of 4; 2range 0 ... 1024 in steps of 4 | |

|  |  |
| --- | --- |
| **Endianness conversion** | |
| REV Rd, Rm | Convert a little endian 32-bit value to big endian, or vice versa |
| REV16 Rd, Rm | Convert an unsigned little endian 16-bit value to big endian, or vice versa |
| REVSH Rd, Rm | Convert a signed little endian 16-bit value to big endian, or vice versa, and sign-extend into the higher 16 bits |

|  |  |  |
| --- | --- | --- |
| **Flow control** | | |
| MOV PC, Rm | | Move any register to PC |
| POP { low\_register\_list, PC } | | Pop the indicated registers and the PC from the stack |
| ADD PC, PC, Rm | | Add low register to PC |
| B label | | Branch to the label |
| B*cond*1 label | | Branch to the label if (and only if) the condition holds |
| BX Rm2 | | Branch to the address in the register |
| 1The conditional branch mnemonics can be one from the next table.  2The lowest bit of the register must be 1. | | |
| BL label | Save PC to LR, then branch to the label | |
| BLX Rm1 | Save PC to LR, then branch to the address in the register | |
| 1The lowest bit of the register must be 1. | | |

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Condition** | **Flags** |
| BCC / BCLO | Carry clear / unsigned lower | !C |
| BCS / BHS | Carry set / unsigned higher or same | C |
| BEQ | Equal | Z |
| BGE | Signed greater than or equal | N == V |
| BGT | Signed greater than | !Z & ( N == V ) |
| BHI | Unsigned higher | C & !Z |
| BLE | Signed less than or equal | Z | ( N != V ) |
| BLS | Unsigned lower or same | !C | Z |
| BLT | Signed less than | N != V |
| BMI | Minus / negative | N |
| BNE | Not equal | !Z |
| BPL | Plus / zero or positive | !N |
| BVC | No overflow | !V |
| BVS | Overflow | V |