Naam:.....richting:.....

Examen Computerarchitectuur en systeemsoftware

Donderdag 15 januari 2009, namiddag

Deel 1: schriftelijk deel

Algemene bemerkingen:

Het examen bestaat uit 2 delen. Dit zijn de vragen voor het eerste deel.

Het eerste deel is schriftelijk en open boek. Open boek betekent dat je alleen je boek mag gebruiken, niet de nota's van de oefenzittingen of opgeloste oefeningen.

Verzorg je schriftelijk antwoord: schrijf leesbaar !

Schrijf bij elke lijn MIPS code commentaar. Dit kan heel kort zijn soms.

Je krijgt maximaal 2 uur voor het eerste deel. Na 2 uur moet je onherroepelijk afgeven. Mocht je vroeger klaar zijn, dan mag je vroeger afgeven. Let op: de vragen moeten mee afgegeven worden. Schrijf op elk blad je naam.

Nadat je dit deel afgegeven hebt breng je je boek weg en krijg je van de surveillant de vragen voor het tweede deel. Het tweede deel is een klassiek mondeling gesloten boek examen met schriftelijke voorbereiding.

Bij elke vraag van dit deel staat een percentage. Dit geeft je een idee van verdeling van de punten over de verschillende onderdelen, en is voor jou een hulp voor de verdeling van je tijd. Weet wel dat de quotering niet een zuivere optel som is van de onderdelen: buitengewoon goede onderdelen leiden tot bonus punten, buitengewoon slechte onderdelen tot negatieve subscores.

Exercise 1 15%

- Give the binary representation of the following decimal numbers: 15, 1492, 269
- Represent the following decimal numbers in 16 bit two's complement: 729, -312, -23, -1
- Convert the following numbers given in 32 bit two's complement binary notation into hexadecimal notation:
  - 1001 1110 1100 1000 0110 0011 1010 0111
  - 0001 1101 0000 0010 1111 0101 0101 0101
- Give the IEEE 754 double precision representation (64 bit) of the decimal number: -387.625

Naam:.....richting:.....

Exercise 2 15%

- Consider a direct-mapped cache with a 5-word block size and a total size of 20 words that is initially empty. Consider the following sequence of addresses, given as word addresses: 29, 63, 75, 39, 13, 0, 4, 22, 66, 12. Reference 0 means the first word in memory, 1 the second, etc. Label each of these addresses as a hit or a miss. Depict the state of the cache at the end of this sequence.
- Consider a 2-way set-associative cache with a 4-word block size and a total size of 24 words that is initially empty. The cache uses the LRU strategy. Consider the following sequence of addresses, given as word addresses: 5, 18, 27, 12, 8, 19, 21, 24, 10, 31. Label each of these addresses as a hit or a miss.
- Consider the 2-way set-associative cache from previous question, in an architecture with 32 bits wide physical addresses and byte-addressing. Calculate the total amount of memory needed to implement this cache (include tag fields, valid bits and data).

Exercise 3 45%

Imagine you're a great cook and just founded your own restaurant. To manage things a bit, you're writing your own C program to store the recipes you invent and their related information (price, time to make ...). The first version of the program is shown below. Because you don't have money yet for expensive compilers, you need to translate the program yourself to MIPS-assembly (you don't have money to buy an Intel-PC either, so you use an old-school SGI Indigo R3000 MIPS workstation to run the program).

- Draw what the program stack looks like at 'POINT A' in the function 'create_recipe' at the first invocation of this function. Local data is stored on the stack. Remember that the program starts its execution with 'main'.
- Write down how the data section of this program will look like in memory (addresses + data) just before program execution. Use this where needed in the rest of this exercise. How many words of the static memory region will be used in total?
- Code the function 'create_recipe' in MIPS-assembly.
- Code the recursive function 'calculate_price' in MIPS-assembly.

You don't have to code the 'main'-routine in MIPS-assembly, nor any other function.

```
/* static data */
static void * top_heap = (void *) 0x1001000; /* current top of heap */
static int current_id = 0;                   /* first not used recipe ID */
static char string[] = "Recipe:";            /* string used in main */

/* recipe structure */
```

Naam:.....richting:.....

```
struct recipe {
        int id;
        char name[15];
        int time_needed;        /* in minutes */
        int ingredients;        /* number of ingredients needed */
        int price;              /* customers pay for the meal */
        struct recipe * next; /* the recipes are arranged in a linked list */
};

/* calculate price (based on time_needed(t) and #ingredients(i)) */
int calculate_price (int t, int i) {

        if (t == i){
                return t;
        }

        return t + calculate_price(t + 1, i);
}

/* alloc function - size in bytes ( allocates memory on the heap) */
void * alloc (int size) {
        int realsize;
        void * temp = top_heap;

        /* always return a word-aligned address */
        realsize = (size + 3) & 0xfffffffc;

        top_heap += realsize;
        return temp;
}

/* create recipe */
struct recipe * create_recipe(char * r_name, int r_time,
                              int r_ingredients, struct recipe * r_list) {

        struct recipe * last = r_list;
        struct recipe * last_min_one = r_list;

        /* POINT A - what does the stack looks like at this point ? */

        /* allocate place on heap for the new recipe */
        struct recipe * r = (struct recipe *) alloc(sizeof(struct recipe));

        if ( r_list != NULL ) {

                last = last->next;

                /* find last element in recipe linked list */
                while ( last != NULL ) {
                        last = last->next;
                        last_min_one = last_min_one->next;
```

Naam:.....richting:.....

```
                }

                /* link the new recipe */
                last_min_one->next = r;
        }

        r->next = NULL;
        r->id = ++current_id;
        r->time_needed = r_time;
        r->ingredients = r_ingredients;

        /* set price customers pay for the meal */
        if ( r->ingredients < r->time_needed ) {
                r->price = calculate_price(r->ingredients, r->time_needed);
        } else {
                r->price = calculate_price(r->time_needed, r->ingredients);
        }

        /* this library function copies the string for you, don't implement it, but call it */
        /* prototype: void strcpy(char to [], char from []) */
        strcpy(r->name, r_name);
        return r;
}


/* main routine */
int main(){

        int i = 0;
        struct recipe * recipe_list_head = NULL;
        struct recipe * r1;

        /* create a recipe */
        r1 = create_recipe("CupOfTee", 5, 3, recipe_list_head);
        recipe_list_head = r1;

        /* create another one */
        struct recipe * r2 = create_recipe("HotChoclate", 10, 2, recipe_list_head);

        /* print recipes */
        while ( recipe_list_head != NULL ) {
                printf("%s␣(nb:␣%d)␣%s␣\n", string, ++i, recipe_list_head->name);
                recipe_list_head = recipe_list_head->next;
        }

        return 0;
}
```

Naam:.....richting:.....

Exercise 4 25%

Write a program in IA32 assembler that reads in a number from a user, converts it to hex and then prints it out again. Remember, read will return a string, you need to convert each string to a number (subtract 0x30 from the ascii value of each digit to get its numeric value). Once you have the number, you can convert it to a hexadecimal string. For example: Reading "42" from a user means you have 2 ascii characters '4' and '2'. You should read the string from the back to the front: the ascii value of '2' is 0x32, subtract 0x30 and you get '2'. Then move on to '4' = 0x34 - 0x30 = 4. Since it's the second digit, you must multiply it by 10. And then add the result to the previous number. You have now converted the number.

Now you can convert the number to hexadecimal in a similar manner, repeatedly divide it by 16. The remainder is the hexadecimal digit. Convert it to a string by adding 0x30 if it's a < 10 or 0x37 if it's $>=10$. You will end up with a string containing the number in reverse.