COMP36512

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Compilers

Date: Tuesday 19th May 2015

Time: 14:00 - 16:00

Please answer any THREE Questions from the FIVE Questions provided

---

This is a CLOSED book examination

The use of electronic calculators is NOT permitted

[PTO]

1. a) For each item in column one, choose the best match for column two. Each item in column two should be used only once.

Column 1	Column 2
1. Dead-code elimination	a. Code generation
2. Interference Graph	b. Code optimisation
3. LR(1) item	c. Context-sensitive analysis
4. Reduce/Reduce Conflict	d. Intermediate representation
5. S-attribute grammar	e. Lexical analysis
6. Sethi-Ullman labelling scheme	f. Parsing
7. Stack frame	g. Parsing
8. Static Single Assignment (SSA)	h. Register allocation
9. Thomson's construction	i. Storage organisation

(5 marks)

b) Assume that you have developed a basic compiler for a subset of the C language that generates code for Pentium single core processors. Identify the components of the compiler that would need to be modified or enhanced to provide the following capabilities:

i) allow C++ (or Java) like comments, that is `//`
ii) add the type `complex` to describe complex numbers
iii) support code generation for a Pentium dual-core processor
iv) include a `repeat ... until` construct. (This construct will repeat the execution of a block of statements until a condition becomes true.)
v) unroll loops as much as possible

Justify your answers. (10 marks, 2 marks each)

c) A certain C compiler performs the following optimisation:

```
/* before optimisation */
for(i=2; i<=n-1; i++)
 for(j=2; j<=m-1; j++)
   a[i][j]=(a[i-1][j]+a[i][j-1]+a[i+1][j]+a[i][j+1])/4;

/* after optimisation */
for(j=4; j<=m+n-2; j++)
 for(i=max(2,j-m+1); i<=min(n-1,j-2); i++)
a[i][j-i]=(a[i-1][j-i]+a[i][j-1-i]+a[i+1][j-i]+a[i][j+1-i])/4;
```

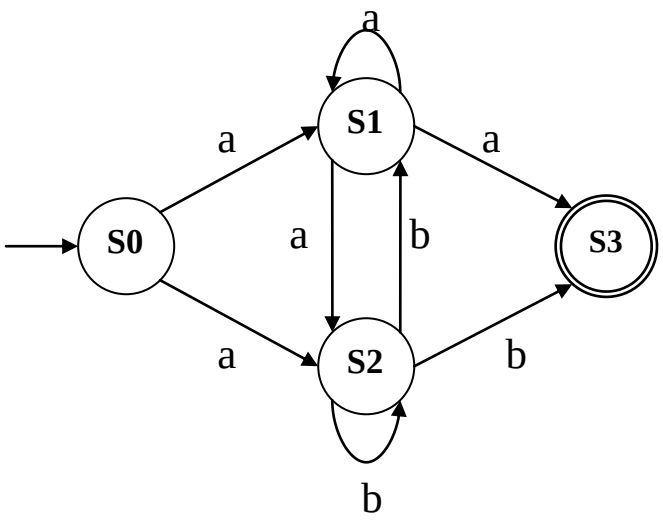
Why is this an optimisation? What is it trying to achieve? Justify your opinion.
(5 marks)

2. a) An integer is divisible by 25 if its last two digits are divisible by 25. For example, 75425 is divisible by 25, while 77865 (and, in fact, any integer ending in 65) is not. Write a regular expression to recognise all integers between 39980 and 99999 which are divisible by 25. (3 marks)

b) Describe the language denoted by the following regular expression and draw the corresponding DFA.

((a | ε) b*)* a (5 marks)

c) Apply the subset construction algorithm to convert the following NFA to a DFA and show the DFA. (6 marks)



d) There are two types of comments in a C++ - like language. The first type considers as a comment everything that follows a pair of forward slashes, //, until the end of the line (\n character). The second type considers as a comment everything which is between /x and the first occurrence of x/, unless this first occurrence is inside double quotes (").

Write a regular expression to recognise comments according to the above. Assume that your input language consists of the five symbols /, x, \n, ", *other* (where the symbol *other* is defined as any character that is not /, x, \n, ").
(6 marks)

[PTO]

3. a) Consider the following context-free grammar:

Goal → L
L → E ; L
L → E
E → E + T
E → T
T → id
T → id ( )
T → id (L)

The terminal symbols of this grammar are: ; + ( ) id

(i) Derive a leftmost derivation for the string

x + y ; z (y ( ))

and show the corresponding parse tree. (5 marks)

(ii) Transform this grammar so that it can be used to construct a top-down predictive parser with one symbol of lookahead. (6 marks)

b) Consider the grammar and the Action and Goto tables below.

1. G → L
2. L → L P
3. L → P
4. P → (P)
5. P → ( )

STATE	ACTION			GOTO	
	(	)	eof	L	P
0	S3			1	2
1	S3		accept		4
2	R3		R3		
3	S6	S7			5
4	R2		R2		
5		S8			
6	S6	S10			9
7	R5		R5		
8	R4		R4		
9		S11			
10		R5			
11		R4			

(i) Show, in full detail, the steps that an LR(1) parser would follow to parse the string (( )) ( ) using the above grammar. For each step, your answer should show the contents of the stack, what the next input is and the action that is taken. (5 marks)

(ii) Consider extending the LR(1) case to the more general cases of using 2 or 3 lookahead symbols. In these cases, for the grammar above, how many columns would the corresponding action/goto table have? Justify your answer. (4 marks)

4. a) Provide semantic rules, using only synthesised attributes, for each rule of the following grammar to calculate the value of the entire expression. The terminal symbol digit may have any value from 0 to 9.

$G \rightarrow E$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow$ digit

State any assumptions you make. (5 marks)

b) A simple language allows program/variable names that consist of one, two, or three characters, with each character being any of the 26 letters of the latin alphabet (this makes it a total of $26+26^2+26^3 = 18278$ possible names). A simple compiler for this language implements a 1024-entry long symbol table. Suggest a good hash function to map program names onto this symbol table. State any assumptions you make. (5 marks)

c) Draw the call graph of the following C-like code fragment.

```
void A(int x) { if(x>0) A(x-1); else B(x); }
void B(int y) { if(y>-2) C(y); }
void C(int z) { printf("%d \n", z); B(z-2); }
main() { A(1); }
```

How many activation records exist in the stack when the execution of this code reaches the `printf` statement for the first time? Your answer should explain what each activation record in the stack corresponds to. (5 marks)

d) Consider the following C-like code fragment.

```
b=4; c=1; d=3; n=5; scanf("%d",&z);
if (d>7) { c+=d+b+z; printf("total %d \n",c); }
q=my_function(z*(c-1));
for(i=10; i<=n; i++)  {  q=q+a[i];  }
printf("final %d \n",q);
```

What kind of code transformation techniques can a compiler apply to improve the efficiency of this code fragment? In each case, give the resulting version of the code after each transformation. Your answer should show what the most optimised version you can obtain is. State your assumptions. (5 marks)

[PTO]

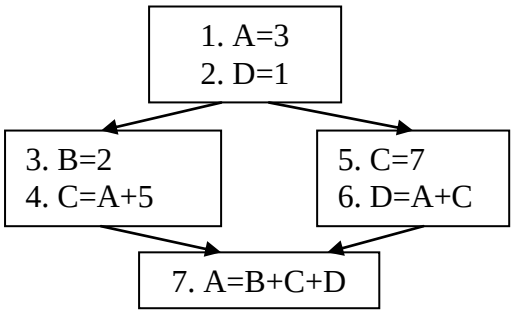
5. a) Consider the following basic block.

```
1. load r1, @x
2. load r2, @y
3. add  r3, r1, r2
4. mult r4, r1, r2
5. add  r5, r3, 1
6. add  r6, r4, r3
7. sub  r7, r6, r4
8. mult r8, r5, r7
```

(i) Identify the live ranges for all register values. (3 marks)

(ii) Draw the interference graph and apply a colouring algorithm of your choice to colour the graph with the smallest number of colours.
(5 marks)

b) Given the following control flow graph and basic blocks, identify the live ranges for all values.



(4 marks)

c) Explain how list scheduling can be used to generate a schedule for the following basic block and show the schedule. First, assume a processor that can issue up to three instructions per cycle, where at most one instruction is a `load`, at most one instruction is a `mult` and at most one instruction is an `add` or a `mov`. All instructions have a latency of one cycle.

```
1. load r1, @x
2. load r2, @y
3. load r3, @z
4. mov  r4, 7
5. add  r5, r4, r1
6. mult r7, r1, r4
7. mult r8, r2, r5
8. add  r9, r8, r3
```

Then, show the schedule for another processor that can issue up to two instructions per cycle (any instructions). Compare the two schedules you derived.
(8 marks)

END OF EXAMINATION