**임베디드 중간고사 답안**

**1.**

;compute and test the condition

ADR r4, x;get address for x

LDR r0, [r4];get value of x

ADR r4, y;get address for y

LDR r1, [r4];get value of y

SUB r0, r0, r1;x-y

MOV r1, #3;

CMP r0, r1;compare

BGE fblock ; if (x-y)>= 3, take branch

;the true block follows

ADR r4, b;get address for b

LDR r0, [r4];get value of b

ADR r4, c;get address for c

LDR r1, [r4];get value of c

SUB r0, r0, r1;compute b-c

ADR r4, a;get address for a

STR r0,[r4];store value of a

MOV r0, #0;

ADR r4, x;get address for x

STR r0,[r4]

B after

;the false block follows

fblock:

ADR r4, c;get address for b

LDR r0, [r4];get value of b

ADR r4, b;get address for c

LDR r1, [r4];get value of c

SUB r0, r0, r1;compute b-c

ADR r4, a;get address for a

STR r0,[r4];store value of a

MOV r0, #0;

ADR r4, y;get address for x

STR r0,[r4]

after : ;code after the if statement

변수를 메모리 주소에서 읽고 저장하지 않았을 경우(-4)

Control 관련 실수(-2)-branch 등

기타 간단한 문법 오류(-1)

**2.**

디스어셈블리 결과 참조

00008390 <foo>:

208 8390: e1a0c00d mov ip, sp

209 8394: e92dd800 stmdb sp!, {fp, ip, lr, pc}

210 8398: e24cb004 sub fp, ip, #4 ; 0x4

211 839c: e24dd008 sub sp, sp, #8 ; 0x8

212 int foo(int x1, int x2)

213 {

214 83a0: e50b0010 str r0, [fp, -#16]

215 83a4: e50b1014 str r1, [fp, -#20]

216 return x1+x2;

217 83a8: e51b3010 ldr r3, [fp, -#16]

218 83ac: e51b2014 ldr r2, [fp, -#20]

219 83b0: e0833002 add r3, r3, r2

220 83b4: e1a00003 mov r0, r3

221 83b8: eaffffff b 83bc <foo+0x2c>

222 }

223 83bc: e91ba800 ldmdb fp, {fp, sp, pc}

224

225 000083c0 <scum>:

226 83c0: e1a0c00d mov ip, sp

227 83c4: e92dd800 stmdb sp!, {fp, ip, lr, pc}

228 83c8: e24cb004 sub fp, ip, #4 ; 0x4

229 83cc: e24dd008 sub sp, sp, #8 ; 0x8

230

231 void scum(int r)

232 {

233 83d0: e50b0010 str r0, [fp, -#16]

234 int i;

235 for(i=0;i<=2;i++)

236 83d4: e3a03000 mov r3, #0 ; 0x0

237 83d8: e50b3014 str r3, [fp, -#20]

238 83dc: e51b3014 ldr r3, [fp, -#20]

239 83e0: e3530002 cmp r3, #2 ; 0x2

240 83e4: da000000 ble 83ec <scum+0x2c>

241 83e8: ea000009 b 8414 <scum+0x54>

242 foo(r+i,5);

243 83ec: e51b3010 ldr r3, [fp, -#16]

244 83f0: e51b2014 ldr r2, [fp, -#20]

245 83f4: e0833002 add r3, r3, r2

246 83f8: e1a00003 mov r0, r3

247 83fc: e3a01005 mov r1, #5 ; 0x5

248 8400: ebffffe2 bl 8390 <foo>

249 8404: e51b3014 ldr r3, [fp, -#20]

250 8408: e2832001 add r2, r3, #1 ; 0x1

251 840c: e50b2014 str r2, [fp, -#20]

252 8410: eafffff1 b 83dc <scum+0x1c>

253 }

254 8414: e91ba800 ldmdb fp, {fp, sp, pc}

256 00008418 <main>:

257 8418: e1a0c00d mov ip, sp

258 841c: e92dd800 stmdb sp!, {fp, ip, lr, pc}

259 8420: e24cb004 sub fp, ip, #4 ; 0x4

260

261

262 int main()

263 {

264 scum(3);

265 8424: e3a00003 mov r0, #3 ; 0x3

266 8428: ebffffe4 bl 83c0 <scum>

267 }

268 842c: e91ba800 ldmdb fp, {fp, sp, pc}

함수 내부의 local var를 문제 1번과 같이 전역변수로 생각하고 ADR 명령어를 통해 변수를 불러 오면(-5) => 일부분을 전역변수와 같이 생각했을 경우(-2)

return값을 전달할 때 r0를 사용하거나 stack을 사용하지 않으면 (-5)

LR값을 저장하지 않고 BL을 중복으로 사용하여 lr값을 알 수 없는 곳으로 jump하게 했을 때(-5)

control 관련 실수(-2)

기타 부분적인 문법 오류(-1)

**3.** 다음과 같이 instruction이 메모리에 들어있다고 가정.

|  |  |
| --- | --- |
| 0 | MOV r0, #0 |
| 1 | LDR r1, #10 |
| 2 | MOV r2, #0 |
| 3 | ADR r3, c |
| 4 | ADR r5, x |
| 5(loop) | CMP r0, r1 |
| 6 | BGE loopend |
| 7 | LDR r4, [r3. r0] |
| 8 | LDR r6, [r5. r0] |
| 9 | MUL r4, r4, r6 |
| 10 | ADD r2, r2, r4 |
| 11 | ADD r0, r0, #1 |
| 12 | B loop |

(a)

초기상태

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. | 초기상태 | 2. | 3까지 수행 | 3. | 7까지 수행 |
| index | Data(instruction addr) | index | Data(instruction addr) | index | Data(instruction addr) |
| 00 |  | 00 | 0 | 00 | 4 |
| 01 |  | 01 | 1 | 01 | 5 |
| 10 |  | 10 | 2 | 10 | 6 |
| 11 |  | 11 | 3 | 11 | 7 |

1. 초기 상태는 empty
2. 3까지 수행됬을 때 0,1,2,3이 차례로 참(cache miss)
3. index에 맞게 4~7이 차례로 참.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4. | 11까지 수행 | 5. | 12까지 수행 | 6. | branch 직후 5~7수행 |
| index | Data(instruction addr) | index | Data(instruction addr) | index | Data(instruction addr) |
| 00 | 8 | 00 | 12 | 00 | 12 |
| 01 | 9 | 01 | 9 | 01 | 5 |
| 10 | 10 | 10 | 10 | 10 | 6 |
| 11 | 11 | 11 | 11 | 11 | 7 |

1. 마찬가지로 8~11이 차례로 찬다.
2. 12는 index가 00 이므로 00에 찬다.
3. branch taget이 5 이고 index는 01 이므로 이 이후에 차례로 찬다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 7. | 8~11 수행 | 8. | 12,5,6,7 수행 | 9. | 종료시 cache 상황 |
| index | Data(instruction addr) | index | Data(instruction addr) | index | Data(instruction addr) |
| 00 | 8 | 00 | 12 | 00 | 12 |
| 01 | 9 | 01 | 5 | 01 | 5 |
| 10 | 10 | 10 | 6 | 10 | 6 |
| 11 | 11 | 11 | 7 | 11 | 11 |

1. 8. 반복해서 7,8과 같은 상황이 된다.(loop)

9. 6(BGE loopend)가 마지막 instr이 된다.

(b)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. | 초기상태 | 2. | 7까지 수행 | 3. | 8~12 수행 |
| index | Data(instruction addr) | index | Data(instruction addr) | index | Data(instruction addr) |
| 000 |  | 000 | 0 | 000 | 8 |
| 001 |  | 001 | 1 | 001 | 9 |
| 010 |  | 010 | 2 | 010 | 10 |
| 011 |  | 011 | 3 | 011 | 11 |
| 100 |  | 100 | 4 | 100 | 12 |
| 101 |  | 101 | 5 | 101 | 5 |
| 110 |  | 110 | 6 | 110 | 6 |
| 111 |  | 111 | 7 | 111 | 7 |

1. empty
2. 순서대로 index에 맞게 들어간다.
3. 8은 index가 0이므로 000 부터 차례로 들어감.

|  |  |  |  |
| --- | --- | --- | --- |
| 4. | branch 후에 5~7수행 | 5. | 종료시 |
| index | Data(instruction addr) | index | Data(instruction addr) |
| 000 | 8 | 000 | 8 |
| 001 | 9 | 001 | 9 |
| 010 | 10 | 010 | 10 |
| 011 | 11 | 011 | 11 |
| 100 | 12 | 100 | 12 |
| 101 | 5 | 101 | 5 |
| 110 | 6 | 110 | 6 |
| 111 | 7 | 111 | 7 |

1. 5는 index가 101 이므로 차례로 7까지 들어감.
2. 종료시 까지 전부 cache hit 이므로 변화 없음.

(c)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | 초기상태 | | | 2. | 0~7 수행 | | |
| SET 0 | | SET 1 | | SET 0 | | SET 1 | |
| index | data | index | data | index | data | index | data |
| 00 |  | 00 |  | 00 | 0 | 00 | 4 |
| 01 |  | 01 |  | 01 | 1 | 01 | 5 |
| 10 |  | 10 |  | 10 | 2 | 10 | 6 |
| 11 |  | 11 |  | 11 | 3 | 11 | 7 |

1. empty
2. 0~7이 차례로 들어간다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3. | 8~11 수행 | | | 4. | 12,5~7 수행(최종 cache 상황) | | |
| SET 0 | | SET 1 | | SET 0 | | SET 1 | |
| index | data | index | data | index | data | index | data |
| 00 | 8 | 00 | 4 | 00 | 8 | 00 | 12 |
| 01 | 9 | 01 | 5 | 01 | 9 | 01 | 5 |
| 10 | 10 | 10 | 6 | 10 | 10 | 10 | 6 |
| 11 | 11 | 11 | 7 | 11 | 11 | 11 | 7 |

1. replacement rule을 어떤 것으로 정하는 가에 따라 과정이 달라진다.  
   여기서 예는 LRU을 사용하여 0~3이 replace 된다.
2. 12에서 branch가 일어나고 5~7은 cache hit가 발생한다. 계속 cache hit이므로 최종 결과도 같다.

기본점수(각 소문항당)

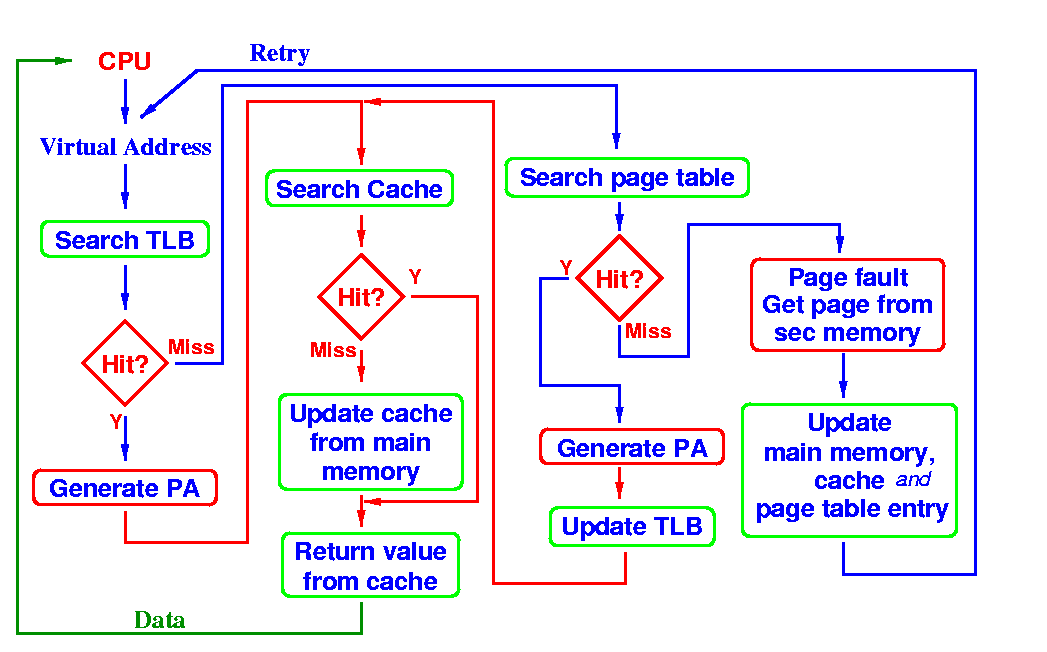
cache에 대한 전반적인 이해가 떨어짐(+2)-> 특히 2-way cache

전체적으로 문제가 없음(+10)

index에 대한 이해가 없어서 replacement가 임의의 위치에서 일어났을 경우(-4)

기타 간단한 오류들의 중요정도에 따라(-1~-2)

**4.**



TLB(+3), cache(+3), page table(+3), 일어날수 있는 상황에 대한 설명(+3)

이며, 각 오류마다 (-1~-2) 씩 감점

**5.**

(a) Compulsory miss : also called cold start miss. A cache miss caused by the first access to a block that has never been in the cache

(b) capacity miss : A cache miss that occurs because the cache, even with full associativity, can not contain all the block needed to satisfy the request.

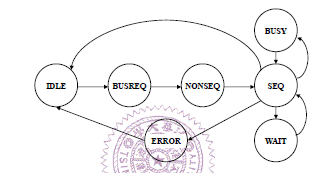
(C) Conflict miss : Also called collision miss. A cache miss that occurs in a set-associative or direct-mapped cache when multiple blocks compete for the same set and that are eliminated in a fully associative ache of the same size.

각 소 문항당 5점씩이며, 전체적으로 잘못된 내용일 경우 1~2점.

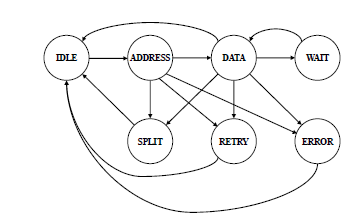
또한 설명에 약간의 오류가 있을 경우 (-1~-2점)

**6.**

(a)



(b)



split, error state는 채점에 고려하지 않음

전체적으로 내용이 맞지 않을 경우(-7)

정확한 signal등에 대한 고려가 거의 없고 상황만을 설명한 경우(-5)

그 외 중요한 state나 진행 방향, cycle이 잘 맞지 않을 경우(-1~-2)

(a)

BURST에 대한 고려가 빠졌을 경우(-3)

REQ state(grant를 대기해야 하는)에 대한 고려가 없을 경우(-3)

(b)

wait, retry에 대한 고려가 없을 경우(각각 -3)