فاز چهارم پروژه کامپایلر – اعلامیه اول

نیم‌سال اول ۸۷-۸۶ – دکتر ابوالحسنی

موعد تحویل: ؟؟/۱۱/۸۶

در این فاز باید برنامه‌ای بنویسید که برنامه‌ی Minijava را به یک زبان میانی شبیه اسمبلی (three address code) که توصیف آن در پایان می‌آید ترجمه کند. البته این کار بعد از ساخت AST و type-checking انجام می‌شود.

خروجی برنامه در حالتی که ورودی مشکل syntax یا lexical دارد باید "syntax error" باشد. اگر برنامه مشکل type دارد، خروجی باید "semantic error" باشد. به بزرگی و کوچکی حروف و فاصله‌ها دقت کنید. خروجی را باید در System.out بنویسید. ورودی را نیز از System.in بخوانید. اگر برنامه درست باشد، باید یک فایل به نام program.asm تولید کنید که کد اسمبلی برنامه در آن نوشته شده‌است.

برنامه‌ی اسمبلی توسط یک شبیه‌ساز که در قسمت resources قرار دارد، تفسیر (interpret) می‌شود. به بزرگی و کوچکی حروف (و همچنین فاصله‌ها) دقت کنید زیرا همه چیز case-sensitive است. به ازای هر گونه بهینه‌سازی در حجم کد و زمان اجرا که انجام دهید و در documentation ذکر کنید، (به نسبت قوت و هوشمندی بهینه‌سازی) نمره‌ی اضافی خواهیدگرفت. documentation در واقع جایگزین تحویل حضوری است، پس آن‌را جامع و مرتب بنویسید.

- sourceهای خود + documentation (با پسوند .txt یا .tex یا .doc یا .pdf) + اطلاعات لازم برای کامپایل و اجرای پروژه: را به صورت proj4-sn1-sn2.zip درآورده (اگر تنها هستید به صورت proj4-sn.zip) و به ce.compiler.course@gmail.com ارسال کنید. Subject نیز باید مشابه اسم فایل باشد، بدون پسوند. پروژه ممکن است تحویل حضوری نیز داشته‌باشد. اگر سوالی داشتید در Discussion Area مطرح کنید یا به abbas.mehrabian@gmail.com بفرستید. به علاوه منظماً به course-page برای تغییرات احتمالی سر بزنید.
- پروژه را باید خودتان انجام دهید، با تقلب به شدت برخورد خواهدشد.

توصیف زبان اسمبلی خروجی

هر برنامه‌ی اسمبلی از تعدادی خط تشکیل شده و در هر خط دقیقاً یک دستور نوشته‌شده‌است. دستورات مجاز عبارتند از:

```
PRINT r-val
        System.out.println(r-val);
ERROR message
        System.out.println(message);
        System.exit(1);

MOV l-val r-val
        l-val = r-val;
AND l-val r-val1 r-val2
        l-val = r-val1 && r-val2;
LES l-val r-val1 r-val2
        l-val = r-val1 < r-val2;
ADD l-val r-val1 r-val2
        l-val = r-val1 + r-val2;
SUB l-val r-val1 r-val2
        l-val = r-val1 - r-val2;
MUL l-val r-val1 r-val2
        l-val = r-val1 * r-val2;
NOT l-val r-val
        l-val = !r-val;
EQU l-val r-val1 r-val2
        l-val = r-val1 == r-val2;

LABEL label
        label:
JUMP label
        goto label;
IF r-val label1 label2
        if (r-val) goto label1; else goto label2;
```

`PARAM` *`r-val`*

Adds r-val to the list of parameters of the function which will be called next

`CALL` *`l-val name count`*

l-val = name(latest *count* parameters that where added using param);

`FUNCTION` *`name`*

indicates the start point of a function

`RETURN` *`r-val`*

return r-val;

`PUSH` *`r-val`*

pushes r-val on the stack

`ALLOCATE` *`l-val r-val`*

allocates r-val words in heap and puts the address of the first word into l-val

`HALT`

System.exit(0);

که زیر هر دستور، معنای تقریبی آن در زبان جاوا نوشته‌شده‌است. در آن‌ها:

l-val ::= HEAP@*atom* | STACK@*offset* | *register*
r-val ::= HEAP@*atom* | STACK@*offset* | *atom*
atom ::= *register* | POP | *int_val* | TRUE | FALSE
register ::= R1 | R2
label, *name* ::= (a-z|A-Z|_)(a-z|A-Z|0-9|_|.)*
message ::= ArrayIndexOutOfBounds | NullPointer
offset, *int_val*, *count* ::= (0-9)*

واحد همه‌ی چیزها در این‌جا کلمه است. هر مقدار عددی یا بولی در یک کلمه جا می‌شود (مقادیر بولی با ۱ و ۰ در حافظه جایگزین می‌شوند)، همه‌ی آدرس‌ها در حافظه هم یک کلمه‌ای‌اند.

به هر تابع یک پشته مجزا اختصاص داده می‌شود. وقتی می‌خواهیم تابعی را فراخوانی کنیم (توجه کنید که تابع در اینجا مفهومی سطح پائین است و شما باید به کمک این توابع، متدها را پیاده‌سازی کنید)، ابتدا پارامترهای آن‌را به کمک تعدادی دستور `PARAM` تعیین می‌کنیم و سپس آن‌را `CALL` می‌کنیم. این پارامترها به صورت پشته‌مانند روی هم قرار می‌گیرند و با هر دستور `CALL` مشخص می‌کنید چند تا پارامتر آخری را باید بردارد و به آن تابع ارسال کند. به مثال زیر توجه کنید:

```
System.out.println(f(1,g(2),3))

f(int a, int b, int c) {
	System.out.println(c);
}
```

می‌تواند ترجمه شود به:

```
PARAM 1
PARAM 2
CALL R1 g 1
PARAM R1
PARAM 3
CALL R1 f 3

FUNCTION f
PRINT STACK@2
```

در بدنه‌ی آن تابع، پارامترها به ترتیب روی STACK قرار دارند. یعنی به اولین پارامتر رد شده، از طریق "STACK 0" دسترسی داریم و الی آخر. به علاوه می‌توانیم هر جا متغیر temp خواستیم از این STACK استفاده کنیم، و نگران قاطی شدن STACKهای توابع مختلف نیستیم، زیرا STACK هر تابع از مال بقیه مجزاست. به علاوه دقت کنید که هنگام صدا زدن یک متد حتماً باید this را به آن رد کنید، زیرا یک تابع، یک موجود سطح پائین است و نمی‌داند که روی کدام object صدا زده شده‌است!

وقتی یک تابع صدا زده می‌شود، قبل از پرش به آن، مقادیر رجیسترها ذخیره می‌شود و وقتی برگشتیم دوباره این مقادیر بازیابی می‌شوند. یعنی تغییراتی که در تابع صدازده‌شده روی رجیسترها ایجاد شود روی تابع صدازننده تاثیری ندارد. خروجی یک تابع را به کمک دستور RETURN برمی‌گردانیم. POP آخرین مقداری که روی پشته PUSH کرده‌ایم را از بالای پشته خارج کرده و برمی‌گرداند. در نتیجه انجام دادن تمام محاسبات به کمک PUSH و POP و دو تا رجسیتر امکان‌پذیر است.

هنگام اجرا یک heap به طول زیاد به طور مشترک در اختیار تمام زیربرنامه‌ها قرار دارد. به کمک دستور ALLOCATE می‌توانید حافظه اختصاص دهید. سپس برای دسترسی به مکان kاُم این حافظه می‌توانید از "HEAP@k" استفاده کنید. مثلاً اگر خواستید مقدار خانه‌ی iاُم یک آرایه‌ی a را چاپ کنید که خود a متغیری محلی است که در خانه‌ی ششم پشته قرار دارد، و مقدار i در حال حاضر روی پشته قرار دارد، یک راهش این است:

```
ADD R1 STACK@6 POP
PRINT HEAP@R1
```

برنامه‌ی اسمبلی باید دو خطای زمان اجرا تولید کند: یکی ArrayIndexOutOfBounds و دیگری NullPointer. این کار از طریق دستور ERROR صورت می‌گیرد پس از رسیدن به دستور ERROR شبیه‌سازی خاتمه می‌یابد (لزومی به نوشتن HALT نیست).

در خط اول برنامه باید دستور FUNCTION *main* قرار بگیرد و آغاز شبیه‌سازی از این خط خواهدبود. در پایان تابع main، باید دستور HALT را بنویسید، که شبیه‌سازی را خاتمه می‌دهد.

دستورات LABEL و FUCNTION ، کاری انجام نمی‌دهند بلکه فقط مکانی را مشخص می‌کنند. پرش به label ای که وجود ندارد یا صدا کردن تابعی که وجود ندارد، باعث بروز Simulation Error می‌شود. به علاوه، label ها و name ها باید در کل برنامه یکتا باشند.

در پایان یک برنامه‌ی کامل و یک ترجمه‌ی ممکن آن آمده‌است. خط‌هایی که با // مشخص شده‌اند برای فهم بیشتر هستند و جزو برنامه نیستند:

```
class Sample {
	public static void main(String[] args) {
		System.out.println(new A().f(5,7));
	}
}
class A {
	boolean x
	boolean s;
	public int f(int a, int b) {
		int ans;
		s = true;
		if (s)
			ans = a – b;
		else
			ans = a + b;
		return ans;
	}
}
```

```
FUNCTION main
ALLOCATE R1 2        //Allocates space for an object of A
PARAM R1             //Passes “this”
PARAM 5
PARAM 7
CALL R1 A.x 3
PRINT R1        //Note that A.x changes the value of R1, but
                //it does not effect value of R1 in main
HALT
```

```
FUNCTION A.x
ADD R1 STACK@0 1    //Load address of this.s into R1
MOV HEAP@R1 TRUE    //Put true into s
ADD R1 STACK@0 1    //Load address of this.s into R1
IF HEAP@R1 L1 L2
LABEL L1
SUB STACK@3 STACK@1 STACK@2
JUMP L3
LABEL L2
ADD STACK@3 STACK@1 STACK@2
JUMP L3
LABEL L3
RETURN STACK@3
```

لطفاً اگر پیشنهاد یا انتقادی درباره‌ی این زبان و امکانات آن دارید به من ای-میل بزنید. به علاوه برنامه‌ی Simulator را نگاه و تست کنید و اگر باگی پیدا کردید حتماً ای-میل بزنید.