Summary of Project 3

12/15/2015

1. Assumptions:

a) For simplicity, when the system runs out of registers, an error will occur. There won't be extra ARM32 code to put it in some kind of stack.

b) GlobalDefs is extended into the FirstPass. However, GlobalDefs is something which usually in the static area, so there is direct way to know their address and offset. So here it is assumed no code generated for the GlobalDefs.

c) Because it always assumes that no more than four parameters in function call, so there is no check for the length of parameters.

2. Some Revisions on the Reference SDD Explanations:

a) Adding some expression (such as E^E, E~E) cases which has been left out.

b) Page4, when enter function call ,the offset is reset to 0 rather than still using x.

c) The initial x is set to -4 rather than 0.

d) Page 8 E[int] - MVN is removed since INTEGER can't be negative and also the negative integer will be parsed as -E, so remove the instruction.

3. Implementation Details

1) Scheme argument explanation. The project avoids the usage of attribute so there are many arguments in a single scheme that needs to be explained. Show below.

#1: The information collected in the first pass, containing DEF Instructions and the other information in NTLT form, which is concatenated using sort PassOne.

#2: Some kind of syntactic sort that needs to generate ARM32 code.

#3: The label generated from outer context, which is used for the inner Statement to jump after finishing.

#4: the offset from the FP. It can be negative or positive. Usually it is negative because it grows towards lower address. But the scheme NegToPos(Computed) will print them as positive numbers.

#5: sort Emaps containing the extended information for Statement in the function members.

#6: The register list containing unused registers.

#7: The IDENTIFIER of the interface used when needed (such as: KVL).

#true:The true label passed down to the conditional expression.

#false: The false label passed down to the conditional expression.

2) Register Strategy

The unused register will be the one that is the next of the current used register. Say, in a situation where Ri is used, the next unused register is Ri+1. So there are several schemes implemented related to the register strategy. In the program, the register is passed down as a register list, containing unused registers in numerical order. So there are both schemes operating on a single register as well as a register list.

a) InitRegList: A scheme that wrap all the registers, representing that at the start of the program there no registers are used.

b) ArgRegList: A scheme that starts with R4, because in function calls previous register might need to store the actual parameters.

c) GetNextReg(Reg): get the next unused register.

d) GetPrevReg(Reg): get the previous used register.

e) GetHeadInRegList(RegList): get the first unused register in the list

f) RemoveHeadInRegList(RegList): means the first one has been used so needs to be removed.

3) SP offset (using FP as base, which is 0). The offset is passed down using built-in sort Computed. Because ARM32 requires the argument to be non negative, so when encounter negative offset, it uses the scheme to transfer it to positive. And the SP is used ADD to represent pointing to higher address, and SUB to lower address.

4) The -, & / are treated the same way as multiplication. So both of their subexpressions are loaded into register first and then used.

5) Type Checker (namely, scheme P2) is always running before running the SecondPass.

4. Limitations

a) The KVL compile scheme generated some errors so the relevant code is cut out to put in the document rather than in the source file. Show below.

b) In the implementations, the conditional expression and the valued expression is separated. So in situation like var b = !a; where !a is a conditional expression, this is not handled in the project because the register of a is not known. However, this can be solved using another helper scheme.

c) The KVL relevant code is directly set to empty Instructions because the some other helper schemes as searching in the information in the PassOne is not implemented due to time limitation.

5. Problems Encountered

1) Page 5: Label can't be converted from computed after concatenation when doing the id_m_entry for function members.

Solution: Adding one rule ⟦ ⟨IDENTIFIER⟩: ¶⟧ to the Instruction below Instruction ⟦ ⟨Label⟩: ¶⟧, which allows computed to be converted to IDENTIFIER and also allows the usage of IDENTIFIER as a single label.

2) Three branches using computed didn't compile. But because of the revision on SDD, so the situation is eliminated.

6. Some Test Result

Test for FirstPass

DEF Instructions:

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='function func (a:string):number {;} inter
face inter {a:string;b:void;}'
  DEF inter_a_offset = 0
    DEF inter_b_offset = 4
    DEF inter__size = 8
```

Test output analysis: Correctly calculated the interface member offset and size of the interface

NTLT to store box MD and the S.uds

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='function func (a:string):number {;} inter
face Inter {a:string;b:void;} var jane:Inter;'
« '$Print2-Pr3Base$Instructions'[
 'Pr3Base$NameTypeListTail_,_NameType__NameTypeListTail_'[
  'Pr3Base$NameType__IDENTIFIER_:_Type_'[
   {"$LineLocation" : STRING[1]; "$ColumnLocation" : STRING[10]}"func".
   'Pr3Base$Type_(_TypeList_)=>_Type_'[
    Pr3Base$TypeList__Type__TypeListTail_[Pr3Base$Type_string, Pr3Base$TypeListTail_],
    Pr3Base$Type_number]],
  'Pr3Base$NameTypeListTail_,_NameType__NameTypeListTail_'[
   'Pr3Base$NameType__IDENTIFIER_:_Type_'["Inter_a_offset", Pr3Base$Type_string],
   'Pr3Base$NameTypeListTail_,_NameType__NameTypeListTail_'[
    'Pr3Base$NameType__IDENTIFIER_:_Type_'["Inter_b_offset", Pr3Base$Type_void],
    'Pr3Base$NameTypeListTail_,_NameType__NameTypeListTail_'[
     'Pr3Base$NameType__IDENTIFIER_:_Type_'[
      {"$LineLocation" : STRING[1]; "$ColumnLocation" : STRING[76]}"jane",
      Pr3Base$Type__IDENTIFIER_[
       {"$LineLocation" : STRING[1]; "$ColumnLocation" : STRING[81]}"Inter"]],
     Pr3Base$NameTypeListTail_]]]],
0] »
```

Test output analysis: Even thought the information is in semantic sort representation, the inner information is correct. The NTLT contains: , func: (string)=>number, Inter_a_offset:string, Inter_b_offset: number, jane: IDENTIFIER,

Test for Valued Expression

E → IDENTIFIER

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='ab;'
  Main:
     MOV R12, SP
     LDR R0,  [ R12,  #4 ]
```

Analysis: output correct.

Test for Valued Expression

E → INTEGER

```
[yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='7;'
  Main:
     MOV R12, SP
     MOV R0, #7

[yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='999;'
  Main:
     MOV R12, SP
     LDR R0,  [ PC,  #0 ]
     B N
    DCI 999
    N:
```

Analysis: output correct. 7 is immediate so directly mov into register while 999 exceeds 255 so it uses DCI to store.

E → STRING

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='"sss";'
  Main:
     MOV R12, SP
     ADD R0, PC, #0
     B N
    DCS "sss"
    N:
```

Analysis: output correct.

E → E(EL)

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='a(2,3,4);'
  Main:
     MOV R12, SP
     LDR R0,  [ R12,  #4 ]
     MOV R4, #2
     MOV R6, #3
     MOV R8, #4
     STMFD SP! , {R0-R3, R12}
     MOV R0, R4
     MOV R1, R5
     MOV R2, R6
     BLX R4
     MOV R0, R0
     LDMFD SP! , {R0-R3, R12}
```

Analysis: Output correct.

Test for Valued Expression

E → E1/E2, E → E1%E2,

```
[yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='2/3; 4%5;'
  Main:
     MOV R12, SP
     B L
     B Ls
```

Analysis: output correct. Since / and % are called from library, so there is no code generated here.

Test for Conditional Expression

E → E1 < E2 (>, <=, >=, ==, !=)

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='5 < 2000;'
  Main:
     MOV R12, SP
     MOV R0, #5
     LDR R1,  [ PC,  #0 ]
     B N
    DCI 2000
    N:
     CMP R0, R1
     BLT L1
     B L2
    L1:
     MOV R0, #1
     B X
    L2:
     MOV R0, #0
    X:
```

Analysis: output correct.

E → INTEGER

```
[yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='7;'
  Main:
     MOV R12, SP
     MOV R0, #7

[yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='999;'
  Main:
     MOV R12, SP
     LDR R0,  [ PC,  #0 ]
     B N
    DCI 999
    N:
```

Analysis: output correct.

Test for Conditional Expression

E → STRING

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='"sss";'
  Main:
     MOV R12, SP
     ADD R0, PC, #0
     B N
    DCS "sss"
    N:
```

Analysis: output correct.

E → E(EL)

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='a(2,3,4);'
  Main:
     MOV R12, SP
     LDR R0,  [ R12,  #4 ]
     MOV R4, #2
     MOV R6, #3
     MOV R8, #4
     STMFD SP! , {R0-R3, R12}
     MOV R0, R4
     MOV R1, R5
     MOV R2, R6
     BLX R4
     MOV R0, R0
     LDMFD SP! , {R0-R3, R12}
```

Analysis: Output correct.

E → E1/E2, E → E1%E2,

```
[yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='2/3; 4%5;'
   Main:
      MOV R12, SP
      B L
      B Ls
```

Analysis: output correct. Since / and % are called from library, so there is no code generated here.

Test for Statement and Statements

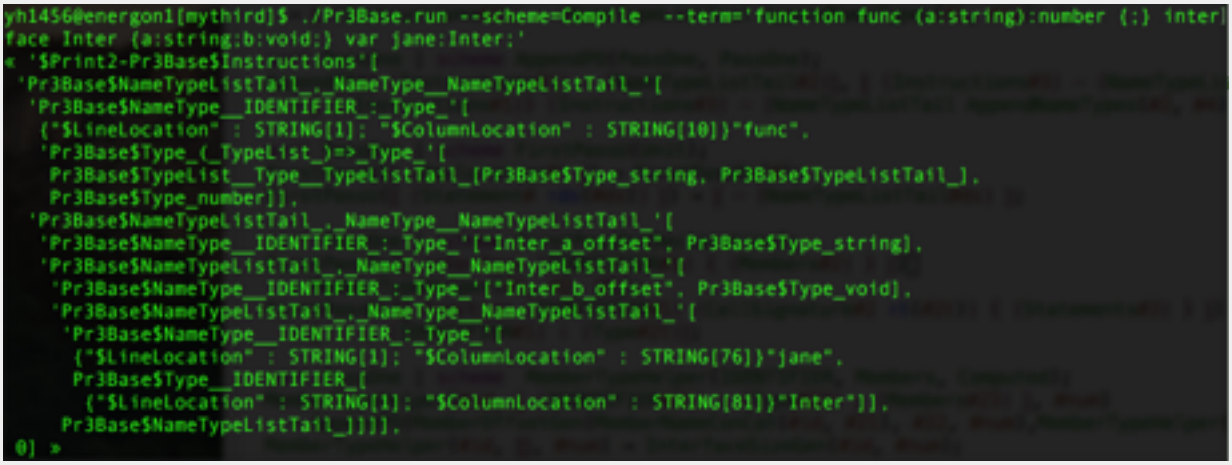
DEF Instructions:

```
yh1456@energon1[mythird]$ ./Pr3Base.run --scheme=Compile  --term='function func (a:string):number {;} inter
face inter {a:string;b:void;}'
  DEF inter_a_offset = 0
    DEF inter_b_offset = 4
    DEF inter__size = 8
```

Test output analysis: Correctly calculated the interface member offset and size of the interface

Test for Statement and Statements

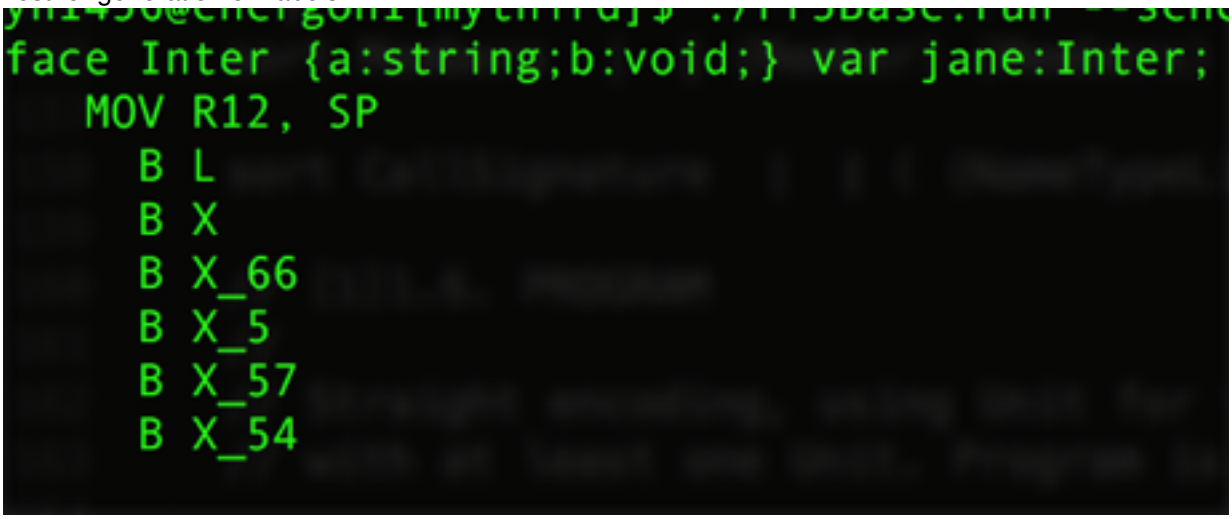
NTLT to store box MD and the S.uds



Test output analysis: Even thought the information is in semantic sort representation, the inner information is correct. The NTLT contains: , func: (string)=>number, Inter_a_offset:string, Inter_b_offset: number, jane: IDENTIFIER,

Test for Unit and Units

Test for generation of Labels:



Test output analysis:correctly output different label for every unit.