CS118 Lab:1

Saw Thinkar Nay Htoo

February 12, 2016

Choosing a car

An algorithm is described to help select the best car based on overall cost related to fuel efficiency.

Inputs: : purchase price and fuel efficiency (in mpg)

Process::

total cost = purchase price + operating cost operating cost = years times annual fuel cost annual fuel cost = price per gallon times the annual fuel consumed.

Output: total cost

analysis

Assuming (since the specification does not indicate it), the number of years will be 10. And then, if the inputs are 15,000 miles per year and gas costs \$3 per gallon. And the inputs are purchase price of \$25,000 and fuel efficiency is 45mpg.

Inputs: : purchase price and fuel efficiency (in mpg)

Process: :

total cost = 25000 + operating costoperating cost = 10 times annual fuel costannual fuel cost = 3 times 15000/45.

Output: total cost

prototype SML code

```
| fun annual_fuel_consumed(ppg,mpy,mpg) = ppg*mpy div mpg;
| val ppg = 3; val mpy = 15000; val mpg = 45;
| val fuel = annual_fuel_consumed(ppg,mpy,mpg);
| val years = 10;
| fun operating_cost(fc) = years * fc;
| val cost = operating_cost(fuel);
| val pp = 25000;
| fun total_cost(oc) = pp + oc;
| val total = total_cost(cost);
```

Output from SML code

```
Poly/ML 5.5.2 Release
val annual_fuel_consumed = fn: int * int * int -> int
val ppg = 3: int
val mpy = 15000: int
val mpg = 45: int
val fuel = 1000: int
val years = 10: int
val operating_cost = fn: int -> int
val cost = 10000: int
val pp = 25000: int
val total_cost = fn: int -> int
val total_cost = fn: int -> int
```

C code implementation of the SML code

```
#include <stdio.h>

| int eax = 0; |

| int annual_fuel_consumed(int ppg, int mpy, int mpg) |
| {return ppg * mpy/mpg;} |

| int fuel; int cost; int total; |

| #define years 10 |

| int operating_cost(int fc) { return years * fc; } |

| #define pp 25000 |

| int total_cost(int oc) { return pp + oc; } |

| #define mpy 15000 |

| #define mpg 45 |

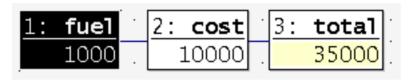
| int fuel; int cost; int total;
```

The output when running C program.

```
Text appended to file lab.c
|int main()
|{
| fuel = annual_fuel_consumed(ppg,mpy,mpg);
| cost = operating_cost(fuel);
| total = total_cost(cost);
| printf("Total cost of car: %i\n",total);
|}
|\clearpage
|\section*{C code implementation of the SML code}
```

```
Text written to file lab.c
|\#include| < stdio.h >
int eax; int ebx; int ecx;
int fuel; int cost; int total;
//int annual_fuel_consumed(int ppg,int mpy,int mpg)
//\{return\ ppg*\ mpy/mpg;\}
void annual_fuel_consumed2()
\{eax = eax * ebx / ecx;\}
# define years 10
int operating_cost(int fc) {return years * fc;}
void\ operating\_cost2()\ \{eax = years * eax;\}
|\#define~pp~25000|
|int\ total\_cost(int\ oc)\ \{return\ pp\ +\ oc;\}
int total\_cost2() \{return pp+ eax;\}
#define ppg 3
\# define mpy 15000
|\#define\ mpg\ 45|
|int main()|
        //fuel = annual\_fuel\_consumed(ppg, mpy, mpg);
        eax = ppg;
        ebx = mpy;
        ecx = mpg;
        fuel = annual\_fuel\_consumed2();
        //cost = operating\_cost(fuel);
                                                   7
        eax = fuel;
        //cost = operating\_cost2();
        operating_cost2(); //assume return value is in eax
        //cost = eax;
```

The output when running the C program in the debugger is:



Translation of C code to ASM code

- include statements (libraries like stdio)
- declare variables
- define functions
- symbolic constants (in C define)
 e.g. #define x 0 in C would be \verb.equ x,0 in asm
- assignment statements
- ullet call assembly language functions
- call C functions (printf)
 e.g. the call to printf: printf("Total cost of car:

```
push %eax\\
push $msg\\
call printf\\
add $8,%esp\\
```

- return from main function
- arithmetic operations (addition, multiplication, integer division)

```
Text written to file lab1.s
equ ppg,3
equ\ mpy, 15000
|.equ~mpg,45|
.data
 fuel: .long 0
 total: .long 0
 cost: .long 0
|msg: .string "fuel: \%i \ n"
.text
 annual\_fuel\_consumed:
 mov $ppg,\%eax
 mov \ mpy,\% ebx
 mul \%ebx \# edx:eax = eax * ebx
 mov \ \$mpg, \%ecx
 div \% ecx \# eax = edx : eax / ecx
 push %eax
 push $msg
 call printf
 add \$8,\% esp
ret
 .globl\ main
|main:
 call\ annual\_fuel\_consumed
ret
```

Shell scripts to make processing easier

This shell script is used to make extracting and processing the source code easier. The -g option to the gcc compiler adds debugging symbols so we can refer to variables even though they are not normally stored in the object code.

```
Text written to file labcode.sh  \begin{vmatrix} docsml & lab1.doc \\ gcc & -Wall & -g & -o & lab1c & lab1.c \end{vmatrix}  Text written to file labdoc.sh  \begin{vmatrix} doctex & lab1.doc \\ pptexenv & pdflatex & lab1.tex \end{vmatrix}  Bourne Shell  \begin{vmatrix} chmod & 755 & labcode.sh \\ chmod & 755 & labdoc.sh \\ \end{vmatrix}   \begin{vmatrix} poly & < lab1.sml > result \end{vmatrix}
```