Десето вежбање

Вежба 1

У овој вежби упознаћемо се са употребом програмских алата намењених за превођење, асемблирање и повезивање програма намењених за ARM архитектуру.

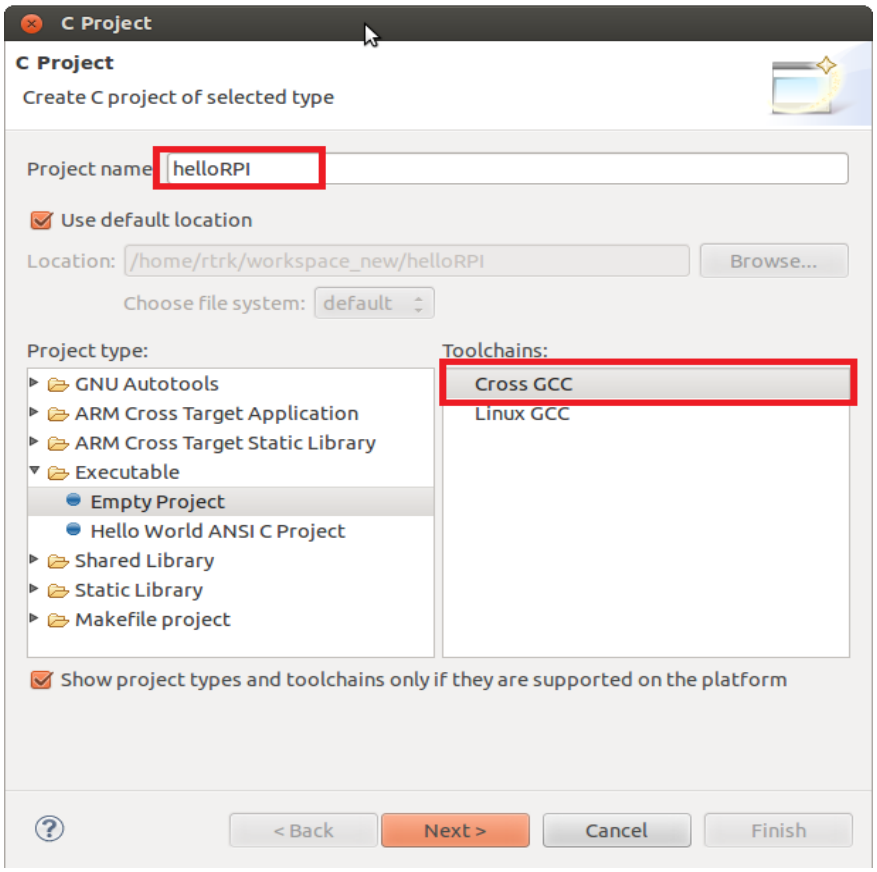
1. Инсталирајте алате за превеођење кода за ARM архитектуру:

   ```
   sudo apt-get update

   sudo apt-get install gcc-arm-linux-gnueabi
   ```

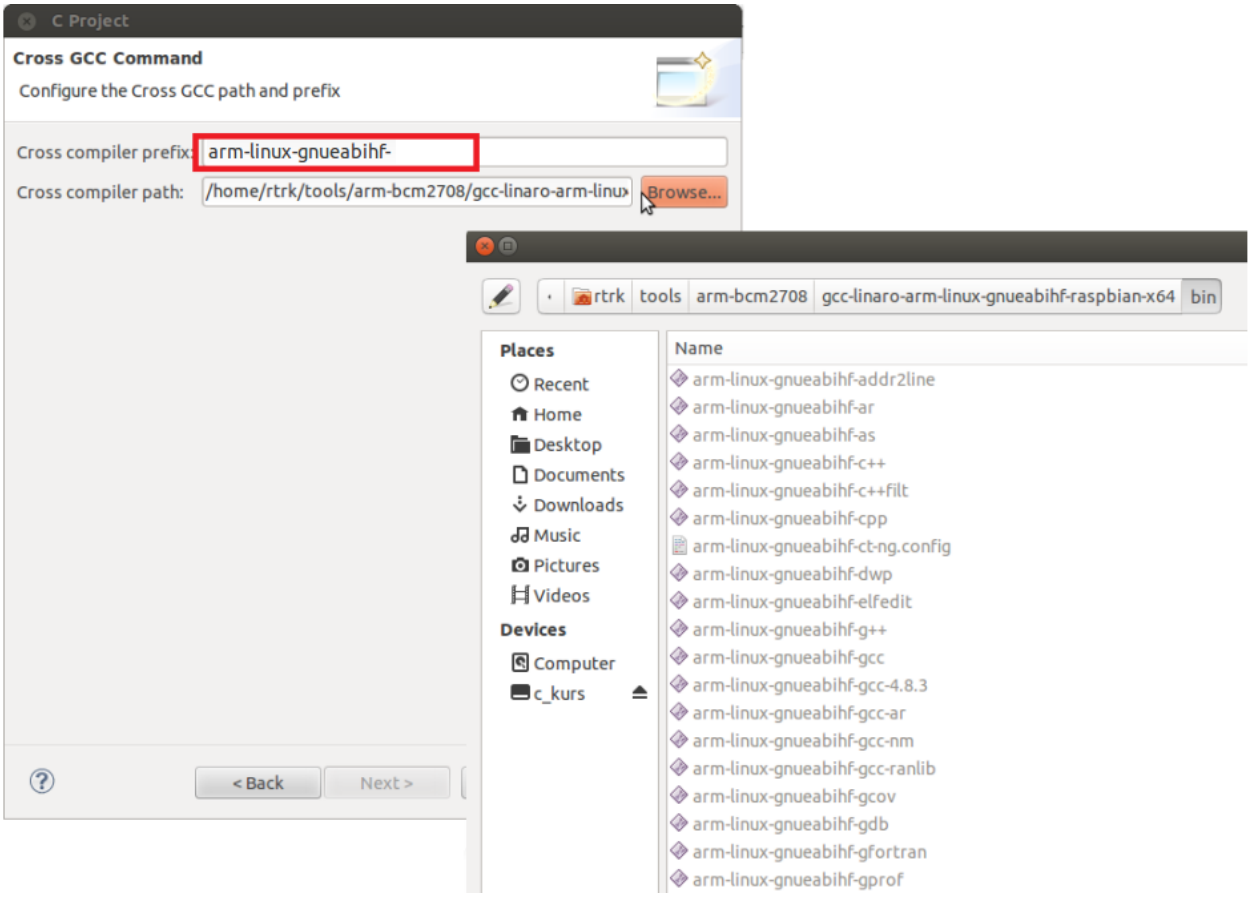
2. Покрените развојно окружење *Eclipse*

3. Направите нови Це пројекат и назовите га „*helloARM*“. Приликом прављења новог пројекта означити да ће бити коришћен скуп алата *Cross GCC*. Дијалог за прављење новог пројекта је приказан на слици испод.



Напредни Це

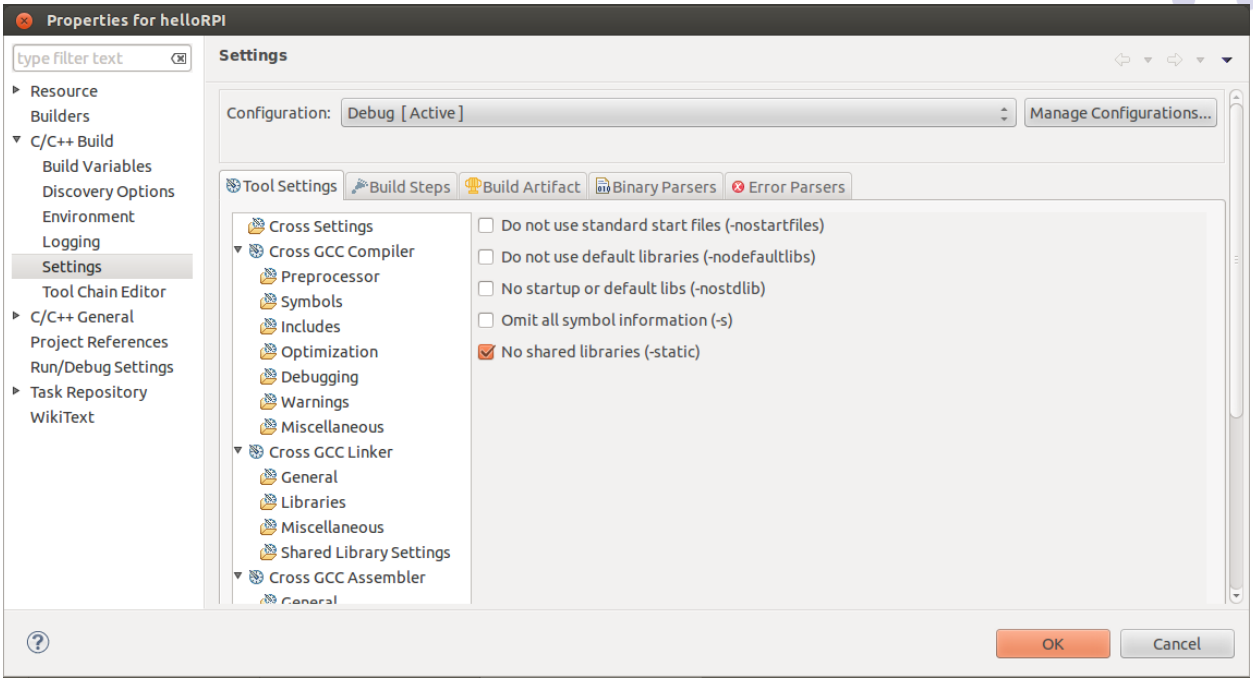
4. У наредном кораку потребно је задати који скуп алата ће бити коришћен:
   - Префикс: arm-linux-gnueabi-
   - Путања до алата: /usr/bin/



5. Додајте нову датотеку у пројекат, са називом *main.c* и у оквору *main* функције додајте испис поруке „*Hello ARM*“.

6. Отворите подешавања пројекта (Project->Propertires). У оквиру подешавања за повезивач (C/C++ Build -> Settings -> Cross GCC Linker), укључите опцију за повезивање коришћењем статичких библиотека (*No shared libraries*)

Напредни Це



7. Преведите програм одабиром команде *Project -> Build*

8. Отворите конзолу и проверите да ли су коришћени програмски алати које сте навели приликом прављења пројекта и да ли је превођење програма извршено успешно.

```
Building file: ../main.c

Invoking: Cross GCC Compiler

arm-linux-gnueabi-gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"main.d" -MT"main.o" -o "main.o" "../main.c"

Finished building: ../main.c


Building target: helloARM

Invoking: Cross GCC Linker

arm-linux-gnueabi-gcc -static -o "helloARM"  ./main.o

Finished building target: helloARM
```

Напредни Це

Вежба 2

У овој вежби ће бити показано на који начин се врши покретање QEMU емулатора и извршавање преведеног програма на емулатору.

- Инсталирајте QEMU (емулатор отвореног кода на којем ће се покретати Линукс кернел)

```
sudo apt-get install qemu
```

- Извршите следећи низ команди како бисте припремили слику фајл система који ће се користити при емулацији.

```
mkdir tmp

qemu-img create rootfs.img 512M

sudo mkfs.ext3 –F rootfs.img

sudo mount –o loop rootfs.img ./tmp

sudo tar xpvf Angstrom-arago-base-image-glibc-ipk-2008.1-test-20090113-beagleboard.rootfs.tar.bz2 –C ./tmp
```

- Копирајте извршну датотеку добијену превођењем кода из претходног задатка у tmp директоријум.

```
sudo cp helloARM ./tmp/home/root/
```

- Покрените емулацију APM платформе у QEMU емулатору (улогујте се као root):

```
qemu-system-arm –M versatilepb –cpu cortex-a8 –kernel ./vmlinuz –hda rootfs.img –m 256 –append “root=/dev/sda mem=256M devtmpfs.mount=0 rw”
```

- Покрените програм у емулатору.

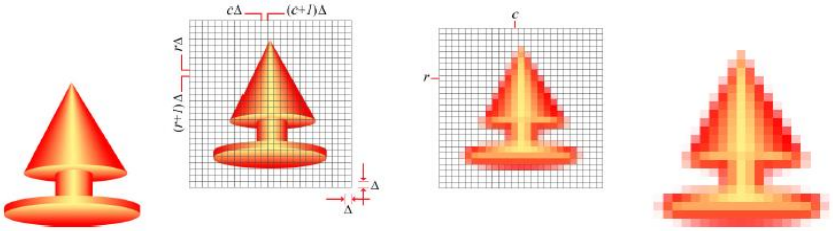
```
./helloARM
```

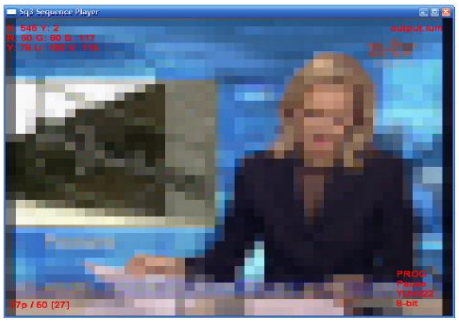
Напредни Це

Вежба 4

У овој вежби је потребно направити референтни пројекат који ће се у наставку вежби користити за проверавање перформанси оптимизација кода коришћењем векторских инструкција и асемблерског кода.

Референтни код садржи имплементацију алгоритма који врши пикселизацију улазне слике. Пикселизација је поступак код ког се из улазне слике узима блок пиксела, вредности пиксела у блоку се усредњавају и тако усредњене вредности се приказују на слици. Примарна употреба пикселизације је цензура.





Пикселизација може да се примењује само на неке делове слике или на целу слику. Алгоритам замућује слику тако што се квадрат од 8x8 тачака попуњава средњом вредношћу тачака из истог квадрата. Дата трансформација се примењује само на Y компоненту.

1. Направите нови C пројекат коришћењем датотека из фолдера *exercises/pikselizacija*. Извршна датотека добијена превођењем и повезивањем овог пројекта треба да се извршава на ARM процесору.

2. Преведите код и извршите га на QEMU емулатору. Потребно је поред извршне датотеке у емулирани фајл систем прекопирати и улазну слику Braid.bmp
3. Погледајте резултат обраде
4. Покрените програм из терминала тако што ћете испред позива програма додати time. Запишите резултате исписане на терминалу

Вежба 5

У овој вежби се уводи оптимизација алгоритма за пикселизацију коришћењем векторских инструкција.

9. Направите нови C пројекат и у пројекат додајте датотеку main.c из директоријума *exercises/pikselizacija_arm_neon*. Прилоком билдовања пројекта је потребно користити *arm-bcm2708-linux-gnueabi* алате.
10. У оквиру подешавања ка компајлер неопходно је додати следеће параметре: **-mfloat-abi=softfp –mfpu=neon**
11. Избилдовати програм и покренути га на QEMU емулатору
12. Проверити излазну слику
13. Покрените програм из терминала тако што ћете испред позива програма додати time. Запишите резултате исписане на терминалу

Вежба 6

У овој вежби је потребно направити пројекат који користи функцију за обраду слике која је написана у асемблеру.

1. Направите нови C пројекат и у пројекат додајте датотек из директоријума *exercises/pikselizacija_arm_neon_asm*. Прилоком билдовања пројекта је потребно користити *arm-bcm2708-linux-gnueabi* алате.
2. У оквиру подешавања ка асемблер неопходно је додати следеће параметар: **–mfpu=neon**

3. Избилдовати програм и покренути га на QEMU емулатору
4. Проверити излазну слику
5. Покрените програм из терминала тако што ћете испред позива програма додати time. Запишите резултате исписане на терминалу

Вежба 7

У овој вежби потребно је подесити механизам за аутоматску проверу усклађености Це кода са MISRA стандардом.

TI компајлер за C6000 породицу процесора нуди могућност провере усклађености кода са MISRA 2004 стандардом (најновија верзија MISRA стандарда је 2012). Обично се за такву врсту провере користе специјализовани алати, или у оквиру компајлера за дату циљну платформу постоји таква могућност. Међутим, GCC не поседује способност такве провере, а тренутно не постоји довољно функционалан специјализовани алат који је бесплатан, или се може без комерцијалне лиценце користити у наставне сврхе. Зато се ослањамо на TI компајлер чију лиценцу имамо. Идеја је да као додатни корак након успешног билдовања програма буде обављено његово поновно превођење TI компајлером са укљученом провером испуњености MISRA 2004 правила.

Ова вежба нема неки посебан крајњи резултат нити проверу, већ служи само да се постави окружење.

1. Прекопирати компајлер за TI C6000 породицу процесора у Ваше окружење.

- Из материјала за вежбу прекопирати садржај директоријума c6000_7.4.23.
- Израз $ТИ_ПУТАЊА означаваће у даљем тексту путању на коју сте прекопирали садржај директоријума

2. Додати у билд процес нов корак након успешног билдовања

- Направити нови пројекат
- У особинама пројекта (Properties) поставити се на следећу путању: C/C++ Build / Settings / Build Steps
- У поље Post-build step / Command унети следећи текст:

```
$ТИ_ПУТАЊА/bin/cl6x --include_path=$ТИ_ПУТАЊА/include --
compile_only --check_misra=all $(C_SRCS)
```

- За потпуно разумевање горње линије погледати у упутству за употребу TI компајлера (spru187u.pdf) поглавља 2.2, 2.3.1 и 6.3.

Вежба 8

У нови пројекат који је направљен у претходној вежби додати датотеку main.c. У ту датотеку ставити садржај program.c датотеке из прве вежбе и покренути билдовање.

Ако је све постављено како треба, конзолни испис ће бити затрпан извештајем о проблемима са MISRA правилима. Усредсредите се на информације које се тичу main.c датотеке (користите претрагу у прозору са конзолним исписом). **Размислите** о сваком нарушеном правилу и о начину како би се оно могло задовољити. За сада није битно да преправите код нити да за свако нарушено правило имате решење.

Вежба 9

Пробајте да преправите код из претходне вежбе, тако да буде задовољено што је могуће више MISRA правила. За разумевање правила (и одлуке како их најбоље задовољити) помоћи ће вам следећи документи:

- MISRA 2004 стандард (misra-c-2004.pdf)
- Списак правила које је дозвољено прекршити и разлог и околности под којима је то дозвољено (MISRA C 2004 Permits.pdf) и
- Табела пресликавања MISRA 2004 правила на MISRA 2012 правила, са образложењима зашто су нека правила промењена (појачана, релаксирана, избачена или су уведена нова) (MISRA C 2012 Addendum 1 - Rule Mapping.pdf)

У MISRA 2004 стандарду погледајте опис сваког нарушеног правила, а онда пробајте да пронађете то правило и у друга два документа.

Посебну пажњу обратите на следећа правила:

2.2 - Ово правило је у новијој верзији стандарда промењено из обавезног у препоручљиво. Анализирајте разлоге за увођење тог правила и за касније избацивање из скупа обавезних правила. Сами одлучите да ли желите да га се придржавате.

5.7 - Избачено у MISRA 2012 стандарду. Размислите о овом правилу и одлучите да ли ћете га се придржавати.

8.1 - Релаксирано у MISRA 2012 стандарду. Анализирајте шта је промењено у овом правилу између два стандарда. (Фраза „internal linkage" означава да је функција приватна)

17.4 - Релаксирано у MISRA 2012 стандарду и промењено из обавезног у препоручљиво. Анализирајте разлике и размислите о њима. Присетите се различитих начина на које можемо декларисати да функција прима низ као параметар - у томе се крије поштовање овог правила.

20.9 - Да ли је ово правило могуће испоштовати у овом задатку?

Попишите сва правила која су иницијално била нарушена и поред сваког наведите како сте га решили.

За одређена правила закључак може бити да не желимо или не можемо да их задовољимо. Коришћењем командне линије или прагми (поглавља 6.3, 6.9.1 и 6.9.25 у упутству за TI C6000 компајлер) обезбедите да анализатор (TI компајлер) изостави проверу тих конкретних правила, на нивоу целог кода или само дела кода. Задатак је готов када анализатор више не буде пријављивао ни једно нарушено правило у main.c датотеци.

Вежба 10

Играјте се мало и анализирајте разне кодове са ранијих вежби. Посебно обратите пажњу на код из претходне вежбе (Структуре података).