

## Lesson 2

السيشن بتاعت النهاردة يا شباب من أهم ال sessions اللي هنتعامل معاها .. يعتبر احنا دلوقتي هنبدأ ال sessions اللي هنخش فيها فعلاً في عمق ال Embedded .. فال session بتاعت النهاردة هي من أهم ال sessions في ال Embedded C هي وبتاعت المرة الجاية .. فركز بقا كدا عشان تيقا ال session ممتعة بالنسبالك وما تشيلش هم ان انت ممكن تقع منك أي حاجة .. انت ان شاء الله هتفهم كل حاجة .

### Booting Sequence

ايه هو ال booting sequence ؟ .. من ضمن الحاجات اللي الناس بتسأل فيها .. بشمهندس هو ال booting sequence بتاع أي بوردة عامل ازاي ؟ .. هو ايه الفرق ما بين ال startup وما بين ال bootloader ؟ .. هو ال startup هو ال processor bootloader ولا لا ؟ .. طب أنا لما باجي أكتب ال application المفروض ايه اللي بيشتغل الأول ؟ .. طب هو ال processor ازاي بيعرف يشغل ال binary بتاعي ؟ .. الأسئلة دي هي اللي بتيجي حوالين الموضوع ده في الانترنت فيوهات .. طب لو انت عندك 2 binary file أو 2 executable image موجودين في ال Flash memory فأنهي واحدة ال processor هيشغلها ؟ .. كل الكلام ده هنجابو عليه في ال section ده اللي هو بتاع ال booting sequence .

---

ايه هو بقا ال booting sequence ؟ .. بصوا يا جماعة .. أول مانت بتفتح ال power بتاع أي بوردة في الدنيا .. ال processor بيخرج على حاجة اسمها ال reset .. أو ال processor ببدا يعمل ال reset state .. ال power ساعات فيه بعض البوردرات بيعملوا ال handle ليها على إنها مش wakeup على إنها reset .. يعني زرار ال reset أو زرار ال power بيبقوا زي بعض في بعض البوردرات .. يعني لو انت كنت شغال ودوست ال reset كإنك بتفتح البوردة من أول وجديد .. كإنك لسا عامل ليها ال power .

---

بيقا لو انت دوست على زرار ال reset في البوردة أو دوست على زرار ال power المفروض بيقا ليهم نفس ال reaction .. طب ايه اللي بيحصل لما بقا البوردة بتقوم ؟ أو ال power بيتفتح ؟ .. يقولك ال processor بيقيم يروح على حاجة اسمها ال entry point .. ايه هي بقا ال entry point ؟ .. ال entry point دي هي بالنسبالك المكان أو ال address اللي موجود في ال Flash memory اللي ال processor لما بيقيم بيشاور عليه .. ده تعريف ال entry point .

---

بيقا ال Flash memory ممكن تبقا بادئة مثلاً من Address 0x00000000 لحد Address 0xFFFFFFFF .. ويقولك بقا ال processor من ال SPECs .. يعني لما تيجي تفتح ال SPECs يقولك البوردة دي ال processor لما بيبجي يقوم منها بيروح عند Address مثلاً 0x100000 .. فال Address ده هيبقا هو ال entry point بتاعت ال processor .. يعني ده ال Address اللي أول ال processor ما يقوم من ال wakeup state أو من ال reset state هيروح عنده .

---

طب هو لما بيروح عند ال Address ده هيلقي ايه ؟ .. المفروض ال processor لما بيروح لأي Address في ال Flash memory يعمل أربع حاجات .. Fetch يعني ايه ال fetch ؟ .. يعني يعمل ال fetch لل assembly اللي في ال Address ده .. طب هو فيه ال assembly بيبقا محطوط في ال Flash memory ؟ .. لا .. امال ايه اللي بيبقا محطوط ؟ .. اللي بيبقا محطوط بيبكون binary .. طب ال binary ده بيتقسم لايه ؟ .. يقولك بيتقسم لحاجتين .. حاجة اسمها ال Opcode وحاجة اسمها ال Operands .. فال Opcode ده اللي بيبكون فيه ال instruction نفسه .. يعني بيبكون فيه ال add أو mov أو load .. يعني ال

opcode ده بيكون فيه ال instruction اللي ال processor بيّفهمه .. هنفهم الكلام ده بعد كدا في الكورس بتاع ال microcontroller architecture .

---

بس خدّها دلوقتي ان ال opcode ده يعني ليه رمز .. ليه binary معين بيعبر عن instruction المفروض ينفذه زي mov وكدا .. طب ايه هي بقا ال operands ؟ .. انت مثلاً لما بتعمل عملية ال add البسيطة دي بتقوله :

add R1, R2, R3

هو بيعمل ايه ؟ هو بيجمع R2, R3 ويحطهم في R1 ده ك assembly .. فين ال opcode هنا ؟ .. ال opcode هو ال add .. وال R1, R2, R3 دول بنسميهم ال operands .. دي ال registers اللي بيكون فيها ال values أو اللي ال add بتتعامل معاها .. هنعرفهم لما ندخل على حاجه اسمها ال ALU بس مش وقتها دلوقتي .

---

تمام .. بيقا أول عملية لما يروح لل address بتاع ال entry point ده هيعمل ايه ؟ .. أول ما يروح لل address ده هيروح يعمل عملية ال fetch .. بعد كدا هيعمل عملية decode .. يعني ايه ؟ .. يعني يفهم ال opcode بتاع ال instruction ده ويفهم ال operands .. وبعد كدا يعمل execute .. يعني ايه ؟ .. يعني يبدأ ينفذ ال instruction ده .. فلو فيه add يجمع ولو فيه load يلوده وهكذا .

---

بعد ما بيعمل بقا fetch و decode و execute .. ببينق لو فيه حاجه المفروض ترجع في ال memory بببدأ يعمل store .. لو مفيش حاجه خلاص .. الكلام ده هنفهمه أكثر في ال Microcontroller architecture .

---

طب ايه بقا حتة ال entry point دي ؟ .. الحتة دي مهمة جداً بقا .. على حسب ال microcontroller هو ببينق فيه .. الفكرة كلها ان عند specific address .. فيه address معين فيها على حسب ال SOC أو ال microcontroller ده .. ال processor أول ما هيقوم هيروح ل Address معين في ال Flash memory ومن هنا يبدأ ينفذ البرنامج بتاعه .. طب أنا ال binary file بتاعي هو فيه ايه ؟ .. ال binary file بتاعي ببينق فيه حاجات كتير .. يعني ببينق فيه sections .

---

ال binary ده ممكن بيقا فيه .txt ( startup.s ) .. ال startup ده المفروض ده ال assembly code أو ده ال piece of code اللي المفروض بتنفذ قبل ال main .. يعني انت المفروض قبل ما تبدأ بتنفذ ال main .. المفروض قبلها بيقا فيه جزء عمل initialization لل hardware ولل processor فالجزء ده بنسميه ال startup ومعظمه ببينق assembly .

---

بعد ال startup ده بببدأ يخش على ال main .. طب الكود بتاعك بقا سواء كان أي C Code بيطلع منه ايه ؟ .. بيطلع منه binary بيعبر عن ال instructions .. بيقولك ال C Code ده أو أي Code في الدنيا ببينق فيه standard section اسمه .txt . ايه ال .txt ده ؟ .. ده convention name ل section اسمه .txt . بيعبر عن أي assembly code موجود في ال program بتاعك .. فمعظم ال assembly code أو ال binary اللي بيعبر عن ال assembly اللي هو ببينق opcode مع operands .. اللي هي الأوامر بتاعتك .. يعني هي كود ال C بتاعك بعد ما اتعمله compile .. بتبقا موجودة في section اسمه ال .txt . واي global أو static variable بيتخزن في حاجه اسمها ال .data . وأي uninitialized global variables بتتخزن في section اسمه ال .bss .

---

ال `startup.s` ده معناه ان لو كله كود بيقا كله هيطلع في ال `.txt`. بس بتاع ال `startup` وبعد كذا كلمة ال `(.txt)` \* .. ال \* دي بتعبر عن `anything` .. يعني أي `instruction` في أي `file.c` عندك .. يعني لو عندك `file.c` 300 فكلهم هيطلع ال `assembly` بتاعهم في ال `.txt`. ده ( خلي بالك ده مش بتاع ال `startup` ده واحد ثاني ) .

---

ال `.data`. يعني كل ال `global variables` بتاعت ال `files` بتاعتك أو ال `static` موجودة في ال `section` ده .. وال `.bss` ده يعني كل ال `variables` ال `uninitialized` هتبقا موجودة في ال `.bss`. سكشن .. وهيبقا معمول ليها `initialization` ب 0 .

---

بيقا أنا في حاجه زي كذا المفروض أخط الكود بتاعي فين ؟ .. المفروض أخط عند ال `entry point` دي أخط عنده ال `startup` بتاعي .. يعني المفروض لما ال `processor` بيحي يقوم ويروح ل `address` معين في ال `memory` يلاقي أول كود يقابله هو كود ال `startup` لأن ده اللي عايزه يتنفذ قبل ال `main` .. بعد كذا بقا يروح لل `main` والأكواد كلها بتاعتك .

---

بيقا دلوقتي نقدر نقول إن ال `processor` ال `first bytes` اللي بيروح يشتغلها بتبقا بعد ال `power` أو إنك تعمل `release` من ال `reset section` .. يعني انت بتروح من ال `reset state` بتاعت ال `processor` يحصل `wakeup` فيبدأ يروح على ال `entry point` .

---

ال `hardware` على حسب ال `SPECs` أو ال `target` اللي انت شغال عليه ببيقا ليه `entry point` بتاعته .. وال `entry point` بتاعته دي المفروض تحط عندها ال `startup code` .

---

لو دخلنا أكثر في الحته دي .. ال `startup` ده `assembly file` .. يعني ممكن ال `startup` ده هو فايل كبير مش صغير .. يعني ممكن أكون معرف فيه حاجات كتير .. ال `assembly` بقا عشان تعمل فيها `function` أو جزء `section` بنعمل ايه ؟ .. بنعمل اسمه `section assembly` .. بنعمل حاجه اسمها `label` :

: label

السطور ال `assembly` اللي هتكتب تحت ال `label` ده هيكونوا مرتبطين بيه .. فكان دول `function` مثلاً .. يعني أنا لو عندي في ال `assembly` فيه سكشن اسمه `reset` وفتحت نقطتين وكتبت سطور `assembly` .. وبعد كذا جيت بعديه وكتبت `section` اسمه `hello` وفتحت نقطتين وكتبت `assembly` .

---

وأنا عايز أقوله روح لل `reset` نفذ السطور اللي فيها .. أعمل ايه ؟ .. أعمل حاجه اسمها `branch` أو `branch label (bl)` .. فده هيروح لل `reset` عشان ينفذ اللي فيها .

---

بيقا بيقولك بقا أهم حاجه في ال `assembly` .. هو انت دلوقتي أول ما ال `processor` يقوم احنا عايزين نحط `section` أو نحط جزء من الكود يتنفذ .. خلي بالك ممكن ال `startup` يكون فيه أجزاء كتير من الكود انت كاتبها .. فأنا عايز أقوله أول جزء عمله في ال `run` هو الجزء اللي اسمه ال `reset section` .. خلي بالك انت ممكن تعمل جزء في ال `startup` باسمك عادي .. يعني مثلاً ممكن تعمل `Keroles_section` وتحط في السكشن ده العمليات الأولى اللي بتتعمل .. بس احنا يا جماعة بنستخدم `convention name` بيكون `already` متعارف عليه في ال `Embedded` .

---

فلما حد بييجي يفتح ال **startup file** بتاعك ويلاقي ان انت عامل **reset section** يقول خلاص بيقا البشمنهندس ده متوقع ان ال **reset section** ده أول حاجة تت **run** .

---

فيه ناس بتعمل كالاتي .. يقولك لأ أنا هستني لما ال **power** يفتح بتاع البوردة أو يحصل **reset** .. ببيقا فيه حاجة اسمها **Exceptional interrupt** .. ببيقا **Reset exceptional interrupt** .. يعني بييجي **interrupt** لل **processor** بيعبر عن ال **reset** .. وببيجوا بقا يعرفوا **section** بال **assembly** إن هو أول ما يروحله يعمل **branch** لل **reset** .

---

المهم يعني إن كلهم بيخلوا أول حاجة ت **run** هي ال **reset section** ده جوا ال **startup** .. طب يا بشمنهندس انت كاتب هنا ان السوفت وير اللي محطوط في البوردة دي .. يا إما ال **bootloader** يا إما ال **bare-metal software** أنا مش فاهمها الحثة دي .. بصوا يا جماعه .. فيه في البوردرات إن أي **processor** في الدنيا ليه **entry point** .. ال **entry address** ده يعني ال **address** اللي هو أول ما يقوم يروحله .

---

ال **bootloader** ده عبارة عن ايه بس قبل ما نتكلم ؟ .. يعني ايه **bootloader** ؟ .. قبل ما نخش في تفاصيله .. قول لنفسك ان ال **bootloader** هو **bare-metal software** .. يعني لما حد يقولك ايه هو ال **bootloader** قوله ده **bare-metal software** بس بيعمل حاجة معينة .. بيكون ليه **startup** و **main** وكل حاجة .. يعني انت تقدر تكتب **bootloader** .. هو مش معجزة يعني .. هو حاجة بسيطة خالص .. واحد عمل **startup** وعمل **main function** وكتب كود بيرمج حاجات في ال **microcontroller** لهدف معين .. ايه هو الهدف ؟ .. هنعرفه بعدين .

---

بيقا ال **bootloader** هو عبارة عن **bare-metal software** .. يعني **software** من غير **operating system** وظيفته يعمل **functionality** معينة .. طب أنا دلوقتي لما هما بييجوا يحطوا ال **loader** ده أو ال **bootloader** ده .. هما بيحطوه عند المكان اللي فيه ال **entry point** ومش بس كدا .. احنا نخلينا أدق بقا .. هما بيحطوا ال **startup** بتاع ال **bootloader** عند ال **entry point** .

---

خلينا أدق أكثر بقا .. هما بيحطوا ال **reset section** اللي في ال **assembly** اللي جوا ال **startup.s** بتاع ال **bootloader** عند ال **entry point** .. يعني اللي يهمني إن أنا أحط ال **reset section** .

---

. First piece of code should be run by the processor at entry point is the reset section

---

أول كود هي **run** محطوط عند ال **entry point** ال **processor** هيقوم عليه .. أنا بقا فاهم كدا .. أنا ليه أحتاج **bootloader** ؟ .. لو البوردة بتاعتي مفهانش **bootloader** .. أنا هحط ال **bare-metal** بتاعي أنا .. يعني لو أنا عملت **bare-metal application** اسمه **Keroles** وظيفته في الحياة إنه لما يقوم ي **toggle LED** .. فأنا ممكن ما أستخدمش **bootloader** .. ليه ؟ .. أنا هكون من ال **SPECs** عرفت ال **processor** أول ما بيقوم بيروح ل **Address** كام .. روجت حظيت ال **reset section** بتاع ال **software** بتاعي اللي هو بي **toggle LED** مثلاً عند ال **entry point** دي .. فلما ال **processor** قام راح عند ال **reset section** بتاعتي ومن جواها عمل حبة حاجات وبعد كدا عمل **branch** لل **main function** ودخل جوا ال **main** اشتغل بقا عادي .. بيقا أنا كدا قدرت أعمل **run** لل **software** من غير **bootloader** .

---

## سؤال انترفيو

بشمهندس هو لازم ال bootloader عشان ت run ال software بتاعك ؟ .. لأ مش لازم .. طالما معاك معلومة ال entry point اللي ال processor بيقوم عليها .. وطالما عندك access ان انت تقدر ت burn your own software عند ال Flash memory وأقدر أحط ال section اللي أنا عايزه عند ال entry point دي فأنا مش محتاج bootloader .

---

أنا همشي معاك واحدة واحدة وبعد كدا هقولك ال bootloader أهميته ايه وفين .. بس احنا الأول بنفهم الدنيا .. بيقا أول حاجة ال software اللي انتوا بتكتبوه يا جماعة يطلع في فايل اسمه ال executable file .. ال executable file ده هو بالنسبالنا دلوقتي هو bare-metal application .. يعني ايه bare-metal application ؟ .. راجع ال lecture اللي فاتت .. ال bare-metal application هو ال application اللي بيكلم ال microcontroller دايركت without any operating system .

---

طب ال executable program/ file اللي انت طلعتة ده هو بيتكون من ايه ؟ .. هو بيتكون من سكاشن .. كل section بيعبر عن حاجة .. هنفهم الكلام ده برضو بعدين ما تقلقش .. أنا باخد ال software بتاعي ده بعمله burn في ال Flash memory يعني بحرقه على ال Flash memory .. يجي ال processor أول ما يقوم يخش على ال entry point يلاقي ال reset section اللي موجودة في ال assembly بتاعت ال startup .. عمل initialization وبعد كدا عمل branch لل main .. ال main هيكون موجود فين ؟ .. في ال .txt .. ليه هيكون موجود في ال .txt ؟ .. لأن ال .txt ده اللي فيه كل الأوامر أو كل ال assembly بتاع كل ال files.c .

---

بيقا لو عايزين ناخذ case study .. بص بقا .. أنا هديك 2 case studies دلوقتي لو فهمتهم انت هتفهم الموضوع ده .. عشان الموضوع ده يا جماعة بيركزوا عليه جداً في الأسئلة بتاعت الانترفيوهات .. فأنا عايزك تفهم .

---

## Case 1

بيقولك ال program بيكون فيه ( program counter register ) PC .. ايه ده ؟ .. ده أحد ال registers اللي موجود في ال processor ووظيفته بيشاور على ال next address اللي المفروض يعملها fetch .. يعني أنا دلوقتي بيشاور على ال reset section ليه ؟ .. لأن ده ال entry point بتاعتي .. هو مش ال reset section هو ال entry point .. لأ .. ده أنا بيشاورك processor على ال Flash memory عند address معين .. ال address ده هو ال entry point .. ال developer وهو بيعمل burn لل software على ال Flash memory راح حطلي عند ال address ده ال reset section أو ال reset vector اللي جوا ال startup .

---

بيقا ايه اللي حصل دلوقتي ؟ .. بيقا based على معلومة ال entry point .. ال developer عمل burn لل reset section في الجزء ده .. لو مكنش ال developer يعرف ال entry point مكنش عرف يحط ال reset section في الحتة دي .. فأول حاجة أول ما بعمل power .. ال program counter هو أصلاً كان بيشاور على ال entry point فأول ما قام راح للمكان ده .. لقي ايه في المكان ده ؟ .. لقي ال reset section .. ال reset section عمل initialization وبعد كدا .. بدأت تعمل الوظائف إن هي تعمل initialization لل processor اللي هي قائمة عليه .

---

بعد كذا بقا تاني وظيفة بيعملها ال **reset section** .. بيروح يجيب ال **.data** . سكشن .. ايه ال **.data** . سكشن اللي في ال **bare-metal SW** بتاعي ده يا جماعة ؟ .. ده ال **section** اللي كان مجمع فيه كل ال **global** وال **static variables** اللي في ال **program** بتاعك .. راح نقله أو عمله **copy** من ال **Flash memory** لل **RAM** .

بيقا تاني حاجه عملها ال **startup** إن هو نقل ال **data section** ده من ال **Flash** وراح وداها لل **RAM** .

---

تالت حاجه بقا راح **حجز مكان** لل **.bss** . ، ايه هو ال **.bss** . ده ؟ .. ده اللي فيه كل ال **variables** ال **global** ال **uninitialized** .. هو حجزلها مكان في ال **RAM** وحطها بأصفار .. طب هل أصلاً ال **.bss** . كان موجود في ال **Flash memory** ؟ .. لأ مكنش موجود .. ليه ؟ .. لأن ال **.bss** . ده يا جماعة .. يقولك إن فيه مثلاً **1KB** المفروض هيقوا أصفار فانا مش محتاج أأخزن حاجه .. فهو كإنه بيحجز مكان لل **.bss** . جوا ال **RAM** عشان بعد كذا ال **software** يروح يكتب عليه .

---

بيقا ملحوظة مهمة .. ال **.bss** . دي مش بتكون موجودة في ال **Flash** .. هي بتتجز في ال **RAM** في تالت خطوة ال **reset section** بيعملها .. لكن هي مش موجودة أصلاً في ال **Flash** .. فهي بيتعملها **Reserve** مش **copy** .

---

بعد ما حجز المكان ده بقا .. راح دخل على المرحلة الرابعة .. ايه هي ؟ .. راح عرّف ال **stack pointer** .. ال **SP** أو ال **stack pointer** ده **register** جوا ال **processor** إن انت بتديله بداية ال **stack** بتاعك .. وبعد كذا هو بقا كل ما بييجي ينده يخش جوا ال **C** كود .. يبدأ ينده على فانكشن بقا .. يبدأ بقا ال **local variables** تتعرف .. وكل فانكشن يت **create** ليها **its own stack** .. ويتعرف ال **variables** فيها .. ولو الفانكشن دي مثلاً ال **main** .. مثلاً ال **stack** بتاع ال **main** .. تيجي ال **main** تنده **function** تانية فتعمل **stack** تاني .. وهكذا .

---

بيقا ال **stack pointer** ده اللي هو فيه بداية ال **stack** اللي بيحدد بيني عليه بعد كذا كل ما يعمل **push** ل **variables** .. طبعاً ال **stack pointer** ده **register** جوا ال **processor** فهو بيديله **initialize** ببداية ال **stack** جوا ال **RAM** .

---

اللي عمل كل الكلام اللي فوق ده هو ال **reset section** .. أو مش ال **reset** بقا .. ال **startup** .. لأن ال **reset section** هو اللي بي **run** عند ال **entry point** .. بس اللي عمل كل الهيصه دي هو ال **startup** .

بعد الخطوة الرابعة دي .. يبدأ ال **startup** بقا يقول أنا خلصت كل المهام بتاعتي .. هروح بقا أنه على ال **main** واللي هي موجودة في ال **main.c** بتاعك ويبدأ بقا يدخل في ال **while(1)** اللي انت بتكون كاتب جواها كل ال **program** بتاعك اللي بيشتغل .

---

بيقا كذا ايه اللي حصل ؟ .. أنا حطيت ال **Bare-metal SW** بتاعي في ال **Flash memory** ( يعني حرقتة عليها ) .. ال **reset section** بتاعي اللي موجود جوا ال **startup** في ال **entry point** .. فال **processor** أول ما قام راحلها .. بدأ ينفذ حاجات من ضمنها إن هو ي **copy** ال **data section** من ال **Flash** لل **RAM** .. بعد كذا حجز مكان في ال **RAM** للجزء بتاع ال **.bss** . سكشن وده هنعرفه بالتفصيل الممل بعدين .. بعد كذا عمل **initialize** لل **stack pointer** اللي موجود في ال **processor** ببداية ال **stack** بتاعي جوا ال **RAM** .. بعد كذا بدأ ي **jump** على ال **main function** بتاعتي .. وال **main**

بتاعتي بعد كدا بدأت تنده على ال ( while ) وجواها بدأ ينده على كل الأكواد اللي كانت موجودة .. وكل الأكواد دي موجودة في ال .txt . سكشن .

لو انت فهمت كدا Case study 1 بيقا فيه فيه .. تعالى بقا نخش على Case study 2 .

## Case 2

بص بقا اللي حصل هنا وركز كويس قوي .. أول ما فتحت الباور للبوردة .. ال processor راح عمل jump على ال entry point .. لقي جواها مين ؟ .. واللي bootloader اللي محطوط هنا مش ال bare-metal بتاعك .. ده أنا هنا حاطط Bootloader.bin وساعات بيسموه ال BootROM .. ال BootROM هو نوع من أنواع ال Bootloader هنفهمها بعدين .. بس هو في الآخر bootloader .. فايه وظيفة ال bootloader والكلام ده ؟ .. هنشوفها دلوقتي .

طب ال bootloader ده عبارة عن ايه ؟ .. هو عبارة عن C كود أو assembly كود عادي خالص .. فهو راح لل reset section اللي موجودة في ال startup بتاع ال bootloader ده .. أول ما راحله عمل ايه ؟ .. عمل نفس الكلام .. ايه هو ؟ .. عمل initialization بعد كدا راح لل main function بتاعت ال bootloader ( عشان بس ما تتلغيطش ) .. جوه بقا ال function بتاعت ال bootloader ده .. جوا ال main بدأ يعمل عمليات .. ايه هي بقا ؟ .. أول عملية وهي أهم عملية .. هو من اسمه كدا Bootloader .. يعني هو بيعمل load لحاجه وبعد كدا ي boot منها .. فيبدأ ال bootloader ده يعمل العمليات بتاعته .. عايزني بقا أ load ليك مين ؟ .. يعني ايه ألود ؟ .. يعني حاجه مش انت بتحرقها at run time .. يعني حاجه بتنتقلها من source ل destination .. في at run time ..

من اسمه كدا بلود .. بلود ايه بقا ؟ .. هديكوا مثال .. فيه عندنا في الويندوز أول مانت بتفتح ال computer ايه اللي بيحصل ؟ .. يجي في ال DOS .. ايه ال DOS ده يا جماعة ؟ .. ال DOS ده في الآخر هو بيشغل حاجه اسمها ال BIOS .. ال BIOS ده نوع من أنواع ال Bootloader .. ال BIOS ده أول ما بيقوم بعد ال DOS .. بيبدأ يروح يلود ال windows image من ال Hard disk .. يعني بيروح لل Hard disk يلود ال windows من ال C partition ويحطها في ال RAM .. وبعد كدا ي run ال software اللي جوا ال RAM .. فال windows يقوم معاك .

يعني يا جماعة .. أنا ممكن أقولكم على فايلات معينة في ال C هي معمولها hidden .. الفايلات دي لو انت مسحتها من ال C partition وقفلت الجهاز وفتحته .. الجهاز بتاعك مش هيشوف ويندوز ومش هيقوم .. الشاشة هتفتح ويقولك أنا مش لاقى ويندوز .. وتقعده تضرب كف على كف .. ازاي .. ده أنا لما خدت ال Hard disk من الجهاز بتاعي وروحت عملته mount عند واحد ثاني .. لقيت ان ال C partition كان مثلاً نص تيرا .. فهما موجودين إلا 12 ميغا مثلاً .. يعني الفايل ال 12 ميغا ده هو اللي عمل المشكلة دي ؟ .. ايوا ما هو الفايل ده اللي فيه المعلومات اللي ال Bootloader لما راح لل C partition كان بيدور على الفايل ده .. عشان الفايل ده فيه معلومات هيقوم ايه من ال C partition عشان يحطه في ال memory فلما انت شيلت الفايل ده ال bootloader مراحش لقاه .. فبالنسباليه مفيش حاجه .. مفيش ويندوز .

يعني من ضمن الحاجات ان كان فيه ولاد في مشروع كانوا بيلعبوا مع بعض .. إن بيخشوا على أجهزة بعضهم يعني وباخدوا الفايلات دي على فلاشه .. ولما البشمةهندس يقعد بقا يلف حوالين نفسه وما يعرفش يجيب الويندوز .. يقولوله هات لنا الهارد بتاعك واحنا نرجعك الفايلات دي عشان الويندوز يتفتح .. طبعاً هزار رخم يعني .

---

اللي أنا عايز أقوله بقا إن ال bootloader ده هو بيروح يلود ال operating system .. ما ال operating system ده يا جماعة هو binary يعني executable .. يعني هديكم مثال .. ال OS بتاع ال Linux .. اللي بنعمله run على Embedded linux board هو عبارة عن إيه ؟ .. 4 ميجا .. ده ال OS نفسه .. بيبقا حاجة اسمها C image أو U image .. وببيقا فيه حاجة اسمها ال UX Kernel وهي دي ال operating sytem .. وهي دي ال Kernel وال Scheduler .. وهي دي اللي فيها كل Network stack وفيها ال device drivers وفيها حاجات كتير قوي .. دي 4 ميجا .. الفايل ده ال bootloader بيروح يلوده من SD Card أو من جهاز موجود متوصل بال Ethernet أو من ال Flash memory .. بيروح يلوده يحطه في ال RAM ويبدأ بقا يعمل run من جوا ال RAM .

---

طب يا بشمهندس أول ما أسمع سيرة bootloader بيقا ده معناه إنه بيلود operating system بس ؟ .. لأ مانت خلاص بقا مخك كبير دلوقتي .. ال operating system هو عبارة عن executable file يعني application .. يعني كإنه هو startup و main وعنده حاجاته بقا .

---

فانت ممكن تلود ال bare-metal SW بتاعك .. يعني انت ممكن تكتب software بتاعك وما يكونش operating system وبرضو تخليه محطوط في حته وال bootloader يقوم ويقوم ال software بتاعك يروح يحطه في ال RAM ويبدأ يشغله .. بيقا مين اللي بيلود ال software بتاعك من مكان 1 .. 2 .. 3 .. 4 .. دي الأماكن المختلفة اللي ممكن ال bootloader يعمل load منها لل software بتاعك .. هنعرف الأماكن دي بعدين .. يعني ال bootloader هو بيلودها من different sources عشان يوديها ل final destination .. طب إيه هي ال final destination ؟ .. هي ال RAM بتاعتك .

---

فبيحط ال operating system أو ال software في ال RAM ويبدأ بقوله execute بقا .. طب لما بيبدا ي execute هو بيعمل إيه ؟ .. هو بيخلي ال processor ي jump على مين ؟ .. على ال reset section اللي موجود في ال software ده .. وفيه بعض ال bootloader بيروح ي jump على ال main علطول على حسب بقا انت مطببط إيه مع إيه .

---

المهم هو ممكن يروح لل startup بتاعك أو يروح ي jump على ال main بتاعك علطول على حسب انت عايز إيه .. بس معظم الناس ال software بتاعها بيكون عنده ال reset section وعنده كل حاجة .. وال bootloader هيروح لل reset section بتاعه يعمل حاجاته وبعد كذا يروح لل main بتاعته ويشغل .

---

بيقا إيه المختلف دلوقتي في ال case study دي ؟ .. أول حاجة نلاحظها إن اللي موجود في ال Flash مش ال bare-metal software بتاعي .. أمال إيه ؟ .. ال bootloader .. طب ال bootloader هو اختراع يعني ؟ .. لأ .. هو عبارة عن bare-metal software بس بينفذ specific function اللي هي ال load و execute وحبة capabilities ممكن نشوفها بعدين .. طب ال processor قام راح لل entry point لقي ال reset section اللي جوا ال startup بتاع ال bootloader راح عمل initialization .. وبعد كذا راح لل data . نقلها كوبي من ال Flash لل RAM .. وراح حجز مكان لل .bss . في ال RAM .. بعد كذا حط بداية ال stack pointer بتاعه في ال RAM .. ليه يا جماعة بيحدد ال stack ؟ .. لأن معظم ال startup بيبقا assembly فمش محتاج stack لأن ما بيكونش فيه local variables في ال assembly .

---

ال local variables دي concept موجود في ال C functions .. فطالما انت شغال assembly بيقا مش محتاجين ال stack .. بس ما ينفعش أروح لل main وأنا مش مطببط ال stack pointer بتاعي .. بيقا أنا كذا بهزر .. فعشان كذا من ضمن



وظائف ال startup إن هو يشاور على بداية ال stack .. وبعد ما يشاور على ال stack .. يعمل الوظيفة رقم 5 بتاعته وهي إنه يروح لل main function .. بيقا دي النقلة اللي اتعملت من ال startup عشان يوصل لل C .

---

بيجي بعد كذا بقا ينفذ ال main اللي موجودة في ال C .. أول ما ينفذ ال main function اللي موجودة في ال C .. بص بقا على اللي هيحصل .. تبدأ ال main تعمل عمليات كتير قوي .. زي ايه ؟ .. انت دلوقتي عايز تلود ال bare-metal software بتاعك أو تلود ال operating system منين ؟ .. هل عايز تلودها من ال Flash memory برا ال microcontroller بتاعك ؟

---

يعني فيه ناس بيقا عندها مثلاً ده الميكروكنترولر أو ده ال SOC وموصل بال EEPROM —> I2C .. ده مشروع هتعملوه ان شاء الله معايا لما نيجي نخش على الميكروكنترولر .. فهو حاطط ايه هنا في ال EEPROM ؟ .. حاطط ال software بتاعك .. اللي هو ال bare-metal SW .. فاحنا عايزين هدفنا ايه ؟ .. نروح نبرمج ال I2C Controller اللي جوا ال SOC عشان نروح نقرا ال software ده كـ byte per byte نقراها ونروح نحطها في ال RAM اللي جوا ال SOC أو اللي جوا الميكروكنترولر .. فبعد كذا يقدر ال processor يتعامل مع ال RAM أسرع من إن هو يتعامل مع ال Flash memory برا ال SOC .

---

تاني الحتة دي .. أنا عندي ( system on-chip ) SOC .. يعني عندي microcontroller فيه CPU وعندي ال Bus وعندي model اسمه I2C Controller وعندي ال RAM .. وال I2C ده واصل ب ال Flash memory .. ده غير ال Flash اللي جوا أصلاً .. أنا عايز أعمل ايه ؟ .. ال bootloader عايز ي initialize ال I2C ده .. يعني ايه ؟ .. يعني يبرمج ال I2C Controller ده عشان يروح يقرا ال data من ال Flash memory يروح يحطها في ال RAM .. عشان بعد كذا ال processor ي jump على ال RAM ويتعامل معاها أسرع بكتير من إن هو يتعامل مع حاجه برا .. بيقا ال software بتاع ال bootloader هيعمل ايه ؟ .. هيعمل ال initialization لل Embedded modules اللي هي هتبقا used في ال loading .. يعني لو هو هيروح يلود حاجه من برا محتاجه ال I2C .. بيقا هيروح يبرمج ال I2C .

---

ال Quadrature SPI model .. QSPI .. ده عشان يوصلوه مع حاجه كذا ال Flash memory ( هنشوف ده بعدين ) .. ولو انت عايز تلود software على سيرفر مثلاً TCP سيرفر موجود على ال Network بتاعتك .. فساعتها هتروح تبرمج ال Ethernet controller .. ولو عايز تبرمج حاجه واصله بال UART هتروح تبرمج ال UART .

---

لو عايز تبرمج حاجه ليها علاقة بال SD Card controller .. يعني انت جيت ال Raspberri أو جيت ال Beagle bone وحطيت جواها ال SD Card وال SD Card بتاعك هو اللي فيه ال Operating system أو فيه Software انت كاتبه .. فانت ال bootloader كذا المفروض هيروح يبرمج ال SD Card Controller عشان يعرف يقرا منها ويطلع منها ال software عشان يحطه في ال RAM .

---

ال GIC هو ال general interrupt controller هنعرفه بعدين .

---

يبقا ده أهم حاجة بيعملها ال bootloader .. وبعد ما بيبرمج أو ي initialize ال modules اللي ليها علاقة بحاجة برا هيروح يعمل حاجة ثانية .. هي load ال software من different sources وده according to the board jumpers . execute it وي RAM .

---

أي بوردة يا جماعة في الدنيا بيبقا فيها jumpers .. ال jumpers دي وصلات انت بتعمل ما بينهم short circuit .. دي بتبقا jumpers أو selectors .. يقولك والله ال bootloader اللي جاي مع البوردة دي .. لو انت عايز تلود ال software بتاعك من ال SD Card خلي ال jumper 1 ده هو اللي متعلم عليه .. يعني ايه ؟ .. يعني روح حط عليه jumper فيبقا انت كدا عملت ايه ؟ .. عملت short circuit ليه .. فيبقا كدا البوردة أول ما ال bootloader يقوم .. ال bootloader ده بيروح بي check على ال value اللي على ال pin دي .. لما يلاقيها إن هي short circuit يعرف إن وصلها Volt معين .. فيعرف من كدا إن هو هيروح لل SD Card controller عشان يطلع ال software من ال SD Card storage يحطها في ال RAM .. بييجي في ال SPECS يقولك لا .. انت لو عايز تلود من ال Flash memory برا ال microcontroller بتاعك واصله بال I2C .. لازم ال jumper 2 هو اللي بيبقا short circuit .. ساعتها تعمل ايه ؟ .. هتقوله خلاص بيبقا أنا هحط ال jumper بتاعي على ال jumper 2 ده .. بيبقا أنا كدا ايه اللي هيحصل ؟ .. هيحصل كالاتي .. ال startup بتاع ال bootloader هيقوم ينده على ال main بتاعت ال bootloader وال main هتبدأ تعمل initialize وتبدأ تقرا .. تلاقي إن ال pin الثانية هي اللي معمول عليها short circuit .. فتقول خلاص بيبقا ده اويشن 2 .. أنا هروح ألود من ال I2C .. بيبقا أنا هروح أبرمج ال I2C وأروح أشوف الفلاشة المتوصلة بيها عشان أقرا .. وهكذا بقا .

---

يبقا لو فيه different booting sequence according to different inputs فاعرف إن فيه أكيد bootROM أو فيه أكيد software .. أو bootloader صغير موجود جوا ال system on-chip دي وظيفته إن هو يقرأ ال jumpers دي و according لل values بتاعتها يبدأ ي configure ال modules اللي هيلود منها ال software بتاعه .

---

تمام .. خلاص أنا عرفت يا بشمهندس من خلال ال jumpers هيقوم منين .. و model ايه وأنا حطيت ال software بتاعي في ال model ده .. لو هو مثلاً ال model ده ال Flash memory واصل بال I2C حطيته .. بييجي بعد كدا يروح لل Flash memory اللي واصله بال I2C ويأخذ ال software بتاعك أو ال Operating system يلوده .. يعني ايه يلوده ؟ .. يعني يعمل ال copy من المكان اللي انت حاطه فيه ويروح ينقله يحطهولك في ال RAM اللي جوا ال Microcontroller .. وانت طبعا سيد العارفين ان ال RAM دي هي في الآخر volatile memory .. يعني once إن ال power قفلت .. كل حاجة على ال RAM اتمسحت .. فالعملية دي بتتكرر مع كل run .

---

كل ال run ال bootloader بيقوم يروح للمكان بجيب ال software يلوده ( يعمل ال copy ) ويحطه في ال RAM .. بمجرد إنه حطه في ال RAM .. يبدأ بعد كدا خلاص بقا شغل ال bootloader خلص .. مع السلامة .. يبدأ يعمل ايه بعد كدا ؟ .. قبل ما ال bootloader يقول مع السلامة يبدأ ي jump .. يعني ايه ي jump ؟ .. يعني يخلي ال program counter register بتاع ال processor يروح ي execute ال startup كود بتاع ال software بتاعك أو بتاع ال operating system بتاعك وبعد كدا ال operating system بتاعك يدخل في ال main وبيبدأ يشتغل بقا وزي الفل .

---

يبقا أنا عايز أقول ايه من الكلمتين دول ؟

عايز أقول ركز قوي .. أول مانت ال startup بتاعك يقوم .. ركز في الحته دي .. انت عشان عندك مكان في ال Stack محجوز فال bootloader ي jump لل reset section بتاع ال software بتاعك .. هل انت محتاج ال Stack وال data . وال bss . بتاعت ال bootloader القديمة ؟ .. ما خلاص انت مش هترجع ثاني لل bootloader انت خلاص روح ل software جديد وبتبدأ ت run .. يعني كأنك كدا ال Bios وداك لل windoes .. هل ال BIOS لسا ال Stack بتاعه موجود

؟ .. لأ خلاص مش محتاجه .. قالك بعد كذا ايه اللي بيحصل ؟ .. ال software بتاعك انت بقا بيدأ يفرد نفسه على إن الحنة اللي انت كنت شغال فيها أثناء ال bootloader بالنسبالي الحنة دي هي reusable memory space .

---

يعني ايه reusable memory space ؟

يعني مكان random في ال memory و can be used عشان يحط فيه ال stack بتاعه .. يعني من الآخر اللي أنا عايز أقوله ايه ؟ .. إن الحنة دي اللي كانت بتستخدم في ال Bootloader من بعد ما ال Bootloader أدى رسالته وإدى العصايله لل reset section بتاع ال software أو ال OS بتاعك .. بقا الجزء من ال bootloader في ال RAM هو بالنسبالي ملهوش لازمه .. reusable memory space .. يعني أقدر أنا ك software بتاعي أستخدمة .. سواء أحطه ك stack بتاعتي .. أشتغل عليه .. أشتغل عليه أي حاجة أنا عايزها وأقدر أحط فيه أي حاجة أنا عايزها .

---

تمام حلو قوي الكلمتين دول .. بيقا احنا عايزين نعرف ايه ؟

من الحاجتين دول لازم تكون فهمت معا ايه هما ال Running Mode .. من Case study 1 و Case study 2 احنا كذا يا جماعه ممكن ن figure out حاجتين .. إن فيه two types من ال Running Mode .. يا إما ال ROM Mode يا إما ال RAM Mode .. ايه هما دول بقا ؟ .. دول 2 concepts معروفين في ال References .

---

ال ROM Mode هو إن ال executable code بتاعك أو ال operating system موجود في ال ROM في ال Non-volatile memory .. وال ROM دي مش ال ROM اللي برا مش ال Flash memory اللي برا ال SOC .. لأ .. في ال Flash memory اللي جوا ال microcontroller وده اللي هو Case study 1 اللي احنا اتكلمنا فيه .. إن انت حطيت ال software بتاعك من غير bootloader .. انت شوفت ال entry point بتاعت ال processor وروحت حطيت ال startup بتاعك في بداية ال entry point ده فانت ال software بتاعك already موجود في ال Flash memory أو ال ROM من غير ما بيقا فيه bootloader أصلاً وبقا استخدام ال RAM بتاعتي بس عشان خاطر أحط ال stack وعشان أ change الداتا اللي هي ال data memory .

---

طب ايه هو ال RAM Mode ؟

ده إن ال software بتاعك ال executable code بتاعك موجود في ال RAM وبيقا في ال ROM فيه bootloader .. بيقا ال ROM ال Non-volatile memory هي فيها ال bootloader اللي هو بيلود ال code image بتاعتك من ال ROM يحطها في ال RAM بعد كذا يشغلها في ال RAM أو يلودها من another location غير ال ROM واللي ممكن بيقا SD Card وممكن بيقا من خلال ال Ethernet وممكن تبقا Flash memory برا الميكروكنترولر .. بس في حالة ال RAM Mode انت ال software بتاعك مش هي run وهو موجود في ال ROM لأن فيه bootloader هو ال reset section بتاعه هو اللي موجود عند ال entry point فلما ال processor يقوم هيروح يقوم ال reset section في ال startup بتاع ال bootloader وال bootloader هو اللي هيروح يلودك من ال ROM أو من another location لل RAM .. وبعد كذا انت بقا تشتغل في ال RAM وتفرّد نفسك في ال RAM وتحط ال Stack بتاعك في ال RAM .

---

طب المزاي ايه ؟ .. يقولك ال ROM Mode هو ال very simple هو ده اللي هنستخدمه في ال STM32 .. لو انت واحد بيكتب Baremetal application هو أنا أحط ليه bootloader أصلاً وأوجع دماغي .. ما أنا أحط ال software بتاعي أحرقة مرة واحدة وأحط ال reset section عند ال entry point وخلصت على كذا .. أنا مش محتاج bootloader أصلاً .

---

ال ROM Mode بي **require smaller memory** .. تخيل كذا أنا مش هحط غير ال executable file بتاعي في ال ROM .. هحط حاجه جنبه ؟ لأ وأشغل ال application بتاعي .. فأنا هاخذ memory صغيرة .

---

ال ROM Mode فيه fixed code addresses .. يعني ايه ؟ .. يعني انت حرقت الكود بتاعك عند ال entry point وده **fixed code address** .. هل ينفع تيجي تاني يوم تحرق فيه ال executable بتاعك تحطه عند **another address** ؟ .. لأ .. لأن ال processor لما بيقوم بيروح لل Address ده .. لو انت حطيتاه عند another address ال processor لما يقوم مش هيعرف يروحله .

---

ال ROM Mode فيه relative small code .. يعني ايه ؟ .. يعني **الأكواد الصغيرة** .

---

طب ال RAM Mode بقا .. يقولك ده Mode Complex .. مانتوا شايفين .. ده فيه bootloader بيقوم .. وال bootloader هو عبارة عن ايه يا جماعة ؟ .. Baremetal software ولكن بينفذ specific function معينة ولما يقوم يبدأ يلّود ال software بتاعك من location لل RAM فتبدأ انت تشتغل مع ال RAM .. فده بنسميه ايه ؟ .. **Relocatable code** ؟ .. ليه relocatable code يا جماعة ؟ .. لأن مثلاً ايه أنا جيت هنا في ال bootloader أول مرة .. قولتله حطلي ال software بتاعي ارميهولي من المكان اللي واخذه منه وروح حطهولي عند Address 0x1000 مثلاً في ال RAM وبعد كذا روح اشتغل .. جيت تاني مرة وقولتله لأ أنا عايزك تحطهولي عند Address 0x2000 وبعد كذا run .

---

ايه ده يعني ايه الكلمتين دول ؟ .. يعني انت ك code مش محطوط عند **Address fixed** .. انت بتتحط عند Address أثناء ال **Loading** .. يعني انت مش معروفك مكان ثابت في ال **memory** .. دا انت ال bootloader بينقلك من المكان اللي انت فيه لمكان تاني على حسب انت دخلتله ايه .. خلي بالك فيه bootloader يا جماعة انت بتديله المعلومات دي على البوردة أثناء ال runtime .. يعني ال bootloader لما بييجي يقوم على ال Rasberri مثلاً أو على ال Beagle bone black انت بتقوله كذا fat load يعني روح لل fat partition اللي جوا ال SD Card لودلي ال software ده من ال SD Card حطهولي عند Address كذا في ال RAM وبعد كذا شغله .. بيقا ال software عرف ال Address بتاعه قبلها ولا أثناء ال Loading ؟ .. أثناء ال loading .. ما دام حاجه لسا معرفتش ال Address بتاعها بنسميها ال **relocatable code** .. يعني code ملهوش fixed addresses .. هو بيتعمله reallocation تبعاً للمعلومات اللي انت بتعملها في ال bootloader .

---

ال RAM Mode ده **very fast** .. ليه ؟ .. معروف يا جماعة ان التعامل مع ال RAM أسرع بكثير من التعامل مع ال ROM .. فده معناه ايه ؟ .. ده معناه إن ال **performance** أحسن .. طب ايه السبب الرئيسي اللي يخليني أمشي بال approach رقم 2 أو case study رقم 2 ؟ .. يقولك لو انت عندك Large code .

---

هتقولي يا بشمهندس بدمتك يعني .. انت لو عندك Large code هتروح تحطه في ال Flash memory وبعد كذا تاخذهولي من ال Flash تحطهولي في ال RAM ! طب ما دام أصلاً ال entry بيتحط في ال Flash ما تشتغل بيه علطول من ال Flash وبلاش bootloader واشتغل ROM Mode .. لأ منا عايز أقولك حاجه .. مقصود إن ال Large code يعني الكود الكبير .. انت مين اللي قالك ان الكود ممكن يكون محطوط في ال ROM ؟ .. هو ال bootloader بيلّوده من different locations ومن ضمن ال locations ال ROM .. بس ممكن أروح ألّوده من SD Card وانتوا عارفين إنني ممكن أوصل SD Card حجمه 64 جيجا بال microcontroller بتاعي عادي خالص .. وممكن أجييه عن طريق ال network .. يعني أحط server

على server وأوصلها ب switch وال switch أوصله بال Rasberri وأروح ألود ال image يعني أجيبها من ال server .. بيقا أنا كدا ماشي بال space بتاع ال server .. فده معناه ايه يا جماعة ؟ .. ان ساعات ال SD RAM بتبقا 2 جيجا لحد 8 جيجا ( خلي بالك في ال ايمبدد عادي : ) وال Flash memory عمرها ما توصل للحجم ده .

---

فانت عندك في بعض الأحيان RAM كبيرة وعندك large code .. بيقا لأ خلي ال large code ده على SD Card وبيقا فيه bootloader مساحته صغيرة ينقلي بس ال software الكبير ده من على ال SD Card يروح يحطهولي في ال RAM الكبيرة وأشتغل علطول من ال RAM .. عشان كذا في ال embedded linux بيستخدموا ال bootloader .

---

تمام حلو قوي .. بيقا احنا كدا يا بشمهندس طلعنا بايه من الحبة دول ؟

إن يا جماعة ما تخافش من حد يقولك ايه ال bootloader تقوم خايف كدا .. لأ يا جماعة ال bootloader ده هو baremetal application ولكن بينفذ specific functions .. ايه ال specific functions ؟ .. بيعمل initialize ال models اللي بت loading ال touch .. تاني حاجه بيلود ال software بتاعك أو ال operating system من different locations لل RAM .. بعد كذا بيعمل execute ليه من ال RAM .. بس .

---

طب يا بشمهندس ال bootloader ده من غير ما ينفعش تشتغل ال binary بتاعك ؟ .. لأ طبعاً .. أنا ممكن أمشي بال approach بتاع ال ROM Mode وأحط my own software ال reset section بتاعه عند ال entry point اللي بيقوم عليها ال processor وبيقا أنا مش عايز ال bootloader خالص في الموضوع .

---

بيجي بعد كذا السؤال اللي ببسأله الناس كثير على النت وفي الانترنت فيوهات وكدا .. يقولك اعلمي comparison ما بين ال bootloader وال startup .. خلي بالك من الحتة دي انت دلوقتي فاهم .. أنا عايزك تبقا فاهم مش عايزك تبقا حافظ حاجه .. هو لا وجه مقارنة أصلاً .

---

ال startup ده assembly code وفيه startup c code هنعرفه بعد شوية .. بس ال startup ده هو ايه ؟ .. هو الكود اللي بي run before ال main .. طب ووظيفته ايه ال startup ؟ .. بي initialize ال processor وبعد كدا من ضمن حاجته إن هو بي copy ال data section من ال ROM يحطها في ال RAM ويحجز مكان لل bss. سكشن .. بعد كدا بي initialize ال stack pointer يعني يحط ال stack pointer بتاعت ال processor يشاور على بداية stack معينة في ال RAM .. يعني انت بتبقا حاططها layout أو في دماغك المكان ده .. بعد كدا بيبدا ي jump لل main .

---

فهو عمل كل الحاجات اللي تخليه يعرف يخش على ال C Code وهو مستريح من غير مشاكل .. هيبدا بعد كدا ال main يشتغل .. فلما تبجي تقول لحد كدا قوله حضرتك أصلاً إن ال bootloader نفسه هو عنده its own startup قوله تعالى نفتح مع بعض على النت نشوف ال U-Boot .. ده نوع من أنواع ال bootloaders اللي بيستخدمها Linux ده عنده its own startup .. أصل انت هتقوله ايه ؟ .. قوله أصل ال bootloader ده مش فيه main function ؟ يعني فيه C ؟ .. ينفع أخش على ال C من غير ما أعمل أهم حاجه ال stack pointer وأعمل ال data section وحاجتها ؟ .. هيقولك لأ .. خلاص ما دام أنا معملتش initialize لل stack هخش على ال main ازاي ؟ بيقا لازم بيقا فيه حاجه تظبطلي الدنيا قبل ما أخش على ال main .. هيقولك اه .. هيقولك بيقا لازم بيقا فيه startup .

---

يبقا قبل ما أدخل على ال main بتاعت ال bootloader لازم يبقا ال bootloader عنده startup ما هو ال bootloader مش اختراع هو في الآخر application .. فال bootloader هو bare-metal application عنده its own startup وظيفته في الحياة إن أنا بحط ال startup بتاعه عند ال entry point بتاعت ال processor .. فال processor يقوم يخش في ال reset section بتاع ال startup بتاع ال bootloader ي jump على ال main .. يبدأ ي initialize ال models اللي هيلود منها ال software الثاني اللي هو بتاعي بقا الأصلي أو ال operating system بتاعي ويحطه في ال RAM ويبدأ بعد كذا يشغله .

---

يبقا ال bootloader بيعمل ايه ؟ .. بيعمل locating و loading وبعد كذا بي passing execution .. يعني بعد ما يخلص خالص دوره في الحياة .. يبدأ بقا يقول لل processor روح للمكان ده في ال RAM ابدأ شغل ال windows أو ال linux أو ال software بتاعي وخلص على كذا .

---

أما ال startup code فهو located inside ال executable file نفسه .. هو ده في الآخر piece of code should be run before main عشان ي prepare ال stack pointer وفيه بعض الحاجات اللي should be used through ال main أو C عامة .

---

يبقا كذا انت فاهم .. أهم حاجه يا جماعة وانت بتتكلم مع حد تبقا انت بتتكلم وانت فاهم .. متخليش حد يضحك عليك .. متخليش حد يقعد يجري ويلف ويدور عليك .. ال embedded سهل الدنيا بسيطة .. هو في الآخر عبارة عن ايه ؟ .. حته assembly مع حبة C بتعملهم compilation يطلعك ال executable file وده فيه حبة سكاشن txt و .data و .bss و debug symbols وحاجات كذا هنشوفها بعدين .. بتاخدها كلها وتبدأ تحط كل section عند مكان معين في ال Flash memory يبدأ تفتح ال power وال processor يقوم عند ال entry point يخش عند المكان الأولاني اللي انت المفروض عامل حسابك ان ده هي ال run الأول يعمل حبة حاجات مهمة وبعد كذا يروح لل main وال main مادام هي C يعني موجودة في ال txt. هيروح للحته دي يبدأ ينفذ حاجاته وخلص على كذا .. ايه اللي فيها ؟ .. الموضوع بسيط وده ال embedded .

---

فمتخليش حد يتوهك بقا .. يقولك والله ال bootloader ده operating system .. من ضمن برضو الأسئلة اللي بسمعها في الانترفيوهات اليومين دول قبل كورونا وكذا .. يقوله يا بشمهندس ال bootloader ده مربوط بال Linux بال Embedded Linux .. يعني ما ينفعش أستخدمه عشان أ run إلا Embedded Linux .. ده True or False ؟ .. نرد عليه نقوله ايه ؟ .. نقوله يا بشمهندس انت بتضحك علينا ! هو ال bootloader ده مش اختراع هو بيتعامل مع operating system as an executable file بياخده من المكان يحطه في ال RAM وبعد كذا يروح ي jump ليه .. فأنا ممكن أحط بدل ال operating system بتاعي my own software .

---

أنا هعمل software بيعت hello على ال UART هحطهولك في ال SD Card وهخلي نفس اللي كان بيعمله على ال Linux بانه يروح على ال SD Card يعملها mount ويخلي ال hello world بتاعتي دي ك executable يروح يحطها في ال RAM ويشغلها فتبدأ ت configure ال UART .. وتبدأ تبعت hello .. بيبقا أنا عرفت أشغله مع baremetal أهو من غير operating system .. خلصت .. يقولك طب ليه الناس ما بتعملش كذا يا بشمهندس ؟ .. بسيطة يا بشمهندس .. لأن الموضوع مش مستاهل bootloader لو أنا عندي software bare-metal مساحته صغيرة .. ليه أعقد نفسي ؟ .. منا أخط ال software بتاعي مكان ال entry point بتاعت ال processor وأخلص نفسي .

---



وعلى فكرة ما ال bootloader هو عبارة عن ايه ؟ .. bare-metal application محطوط ال startup عند ال entry point وخلصت على كذا .. ما تخليش حد يضحك عليك .

---

قبل ما ندخل بقا على ال concept ال bootloader in depth بقا .. احنا كذا خدنا الدنيا كذا بس بنسخن .. سيشن النهاردة ممتعة للي هيفهم .. هتبقا مش ممتعة ومقرفة وهنتضايق منها لو انت مش فاهم .

---

يبقا كل اللي فات ده انت فهمت معنى ال bootloader .. كل اللي فات ده فهمنا ان فيه حاجتين .. فيه BootROM وفيه BootRAM وفهمنا ان فيه Case studies 2 ان انا ممكن أحط my own executable دايركت على ال entry point بتاعت ال processor .. وفيه Case study تانية هي ان انا أحط bootloader يقوم وبعد كذا يقوم ال software بتاعي .

---

## (Bootloaders in depth (real cases using OS

نيجي بقا دلوقتي ن focus ونخش جوا ال bootloader ده .. هنلاقي ان ال bootloader ده جواه أنواع .. هو مش حاجه واحدة .. فيه أنواع لل bootloader وفيه كذا case study جوا ال bootloader نفسه .. ايه هي الأنواع دي بقا ؟ .. عشان نفهم الأنواع تعالى بقا نعلي المستوى شوية .

---

تعالى نشوف بوردرات حقيقية .. هنلاقي فيها ايه ؟ .. بص بقا فيه processor .. انت بقا أنا عايزك في ال ايمبدد اعتبر ال embedded ده عبارة عن عدة .. من دلوقتي لحد لما نخلص ال Mastering embedded system online diploma دي اعتبر أي System on-chip في الدنيا أو أي microcontroller .. الاتنين equivalent لبعض بس ال on-chip هو high performance microcontroller .. أي واحدة فيهم هي في الآخر عبارة عن ايه ؟ .. chip كذا حلوة جميلة طالعة منها pins .

---

ال chip الحلوة الجميلة دي لما أجي أفتحها من جوا هلاقيها عبارة عن كذا .. هلاقيها عبارة عن processor وفيه bus وال bus ده أنواع AXI, AHB وحاجات كذا كتير قوي هنعرفها لما نيجي نخش في ال embedded .. وال bus ده بقا متوصل بمين ؟ .. ب memory .. ممكن فيه بعض ال microcontrollers يقولك ال processor واصل ب code flash و data flash و SRAM1 و SRAM2 و SRAM3 .. هما دول عبارة عن ايه ؟ .. عبارة عن 2 category واحدة ROM وواحدة RAM فمتخفش من الأسامي .

---

فيه بعض ال microcontrollers يقولك لا لأ .. أنا ال microcontrollers بتاعي according to the SPECS معندوش إلا Flash memory واحدة وعندها SRAM واحدة .. فيه بعض ال microcontrollers التانية يقولك أنا عندي Flash memory واحد وعندي Code flash وعندي ال RAM وعندي QSPI Model متوصل بقا برا ال microcontroller خالص ب Flash memory تانية كبيرة هقوله ايه ؟ .. هقوله مع احترامي ليك .. انت في الآخر خالص عندك ايه ؟ .. عندك Flash واحد فوق و Code Flash وعندك Flash برا ال microcontroller .. بيقا انت عندك كام Flash حضرتك ؟ .. 3 .. دا غير ال SRAM اللي جوا اللي هي نوع الرامات .. يعني انت في الآخر عندك ايه ؟ .. عندك 3 ميموري تحت ال category بتاع ال Flash وعندك 1 ميموري تحت ال Category بتاع ال RAM ما تضحكش علينا .. يعني يا جماعة مفيش microcontroller عنده قاعدة ثابتة .

---

لأ ده كل حد عمل design أو كل شركة semi-conductor عملت microcontroller هي عاملة زي ال puzzle هي بتحط على ال bus الشكل بتاعها ال system بتاعها .. فمتأخدش قاعدة إن ال microcontroller لازم يكون فيه code flash .. ال microcontroller ممكن متعرفش يعني ايه code flash .. يعني ايه code flash ؟ هو Flash memory .. بس هما مسمينه code flash عشان يقول لحضرتك لما تيجي تحرق الكود بتاعك بيقا حطه في ال memory دي احنا مسمينهالك أهي code flash عشان تستخدمها لحرق الكود .. بس أنا ممكن أحرقه في أي حته .. مين الفواصل ما بيننا ؟ .. ال entry point بتاعت ال processor ده .

---

طب هل ينفع ال processor ده بيقا ليه multiple entry point ؟ .. اه ينفع بيقا ليه multiple entry points .. طب ازاي ؟ .. على حسب ال jumpers بقا .. مش احنا قولنا دلوقتي إن اللي بيتحكم في الموضوع ده انت بتروح لفين هو ال jumpers ؟ .. هتقولي طب ازاي في ال processor بغير ال entry point بتاعته بال jumpers ؟ .. لأ هو ال processor ليه entry point واحدة بس أكيد بيروح ل bootloader ال bootloader ده بيقرأ ال jumper وعلى أساسه ببدا يروح يشتغل عند كذا .

---

بيقا خلاص حته اللي على ال bus دي عندك ايه دي بتاعت ال microcontroller .. تعالوا نشوف المثال ده هو عنده ايه ؟ .. عنده NOR Flash وعندك SD Card controller ودي embedded module ليه registers و انت تقدر تبرمج وليه Drivers ودي واصله بال SAN Disk اللي هي ال SD Card اللي انتوا عارفينها .. وعندك DRAM Controller واصل ب DATA وده واصل بحاجه اسمها DFI PHY ده نوع من أنواع ال protocols PHY هنعرف الكلام ده في ال embedded .. المهم يعني ان فيه حاجتين يا جماعة بيتحكموا في حاجه اسمها ال DRAM .. بيقا ال DRAM دي هي رامة عادي واصله بال bus بس عشان يعرف يكتب عليها فهي متوصلة ب 2 controllers هنعرفهم بعدين .

---

وعندي Ethernet controller واصل بالانترنت ومن خلال الانترنت هو واصل ب server كمبيوتر تاني بالشبكة عندي في البيت محطوط عليه DHCP Server .. تمام تقولي ايه من الكلمتين دول ؟ .. أول حاجه أنا رocht حظيت ال software بتاعي وهنا السوفت وير بتاعي اللي أنا مهتم بيه هو Operating System وال OS ده أنا مهتم بال Linux Kernel .. يعني أنا هنا ال software اللي أنا عايز أعمل run ليه هو ال Linux Kernel ده فده محطوط في 4 ميغا في فايل اسمه ال ulmage بيقا ال uimage ده عبارة عن ايه ؟ .. هو binary code لل linux operating system أو ال zimage .. طب ايه الفرق ما بين ال z image وال u image ؟ يقولك ال z image هي ال u image اللي هي ال operating system نفسه ولكن معمولها zipping يعني هي ال zip version من ال u image يعني ال compress version من ال u image .. يعني كأنك خدت ال u image عملتها ضغط ضغطتها وحطيتها على ال SD Card بيقا انت يا إما تحطلي دي يا إما تحطلي دي على ال SD Card .

---

ال bootloader هو responsible إن هو يعمل ايه ؟ .. Basic hardware initialization .. طب هو فين ال bootloader حضرتك ؟ .. ال bootloader رocht حظيته في ال NOR Flash وحطيت كمان في ال NOR Flash دي ال Kernel بتاعت ال Linux اللي هي ال u image أو ال z image وظبطت إن ال bootloader ده ال reset section بتاعته تبقى موجودة عند ال entry point بتاعت ال processor .. فايه اللي حصل بقا ؟ .. أول ما فتحت البوردة ال processor راح لل entry point بتاعت ال bootloader عمل ال basic hardware initialization وبعد كذا راح لود ال application بتاعك .. ال application بتاعك ده يا إما هو ال software بتاعك ك binary يا إما وده ال usually أو ال most famous أو ال most used إنه يكون ال operating system kernel بتاعك .

---



عندنا بقا مش أنا بقولك هو بيروح يلّوده من different sources يوديه ل destination معين .. ففي الحالة اللي أنا بشرحها دي هو راح يلّوده من ال Flash storage يعني ايه ؟ .. يعني أنا كنت حاطط ال Linux kernel بتاعتي في ال Flash memory فهو ال bootloader بعد ما اشتغل اديته أمر إنه يلّودلي ال software بتاعي اللي هو ال Linux kernel من ال Flash memory يعملها copy ويروح بيها يحطها فين ؟ .. بص كدا بيروح يحطها في ال RAM أو ال DRAM ليه ؟ .. لأن ال RAM أسرع وأكبر وبعد ما حطها في ال الذاكرة ال processor يبدأ بقا من على ال bus يروح لل RAM ويبدأ يعمل fetch decode execute لل assembly اللي فيها فيبدأ يشغل ال operating system أو ال software بتاعك .

---

ممكن يعمل ايه تاني ال bootloader ؟ .. ممكن يلّود from the network .. يعني ايه ؟ يعني أنا اديته أمر إنه يروح يجيبلي ال kernel نفسها أو ال software بتاعي من الكمبيوتر اللي واصل بال switch وال switch واصل بال Ethernet jack اللي هو واصله بال Rasperri bi طب هو عمل كدا ازاي ؟ .. هو ال bootloader لما اشتغل يا جماعه راح خلى ال processor يكتب على ال register بتاع ال ethernet controller فال ethernet controller اتعمله initialization فبدأ يقدر يتعرف على الشبكة من خلال ال switch فيبدأ أنا اديته أمر إنه يروح لل server ده بال ID Address ويجب منه ال file يعمل copy ويروح يحطه فين ؟ .. في ال RAM وبعد كدا يروح يشغله من ال RAM .

---

لا ده كدا الموضوع جميل قوي .. أقولك تعالى نشوف اللي بعده .. ال kernel لو موجود في SD Card .. يعني هنا ال bootloader بعد ما اشتغل خلى ال processor يروح يكتب على ال registers بتاعت ال SD Card Controller عشان ال SD Card Controller يقرأ ال software بتاعك أو ال OS ينقله من ال SD Card ويروح يحطه في ال RAM .. وبعد ما يحطه في ال RAM يعمل ايه حضرتك ؟ يشغله .. طيب حلو قوي .

---

ال bootloader بيعمل ايه تاني ؟ .. بجانب كل الحاجات دي فيه بعض ال bootloaders بيدوك interaction .. يعني ايه ؟ .. يعني مثلاً حاجه زي ال U-Boot ده نوع من أنواع ال bootloaders ده بيخلي ال microcontroller بتاعك من خلال ال UART يوصله عن طريق ال USB بتاع ال computer بتاعك فيبتدر تفتح ال terminal ال terminal دي هي مفتوحة على الكمبيوتر بتاعك ولكن هي بتكلم عن طريق ال UART ال microcontroller اللي عليه ال bootloader فتدي أوامر لل bootloader at runtime .. يعني انت تقول لل bootloader كدا تنيله أمر على ال terminal تقوله روح لودلي من SD Card أو روح لودلي من Ethernet أو روح لودلي من ال Flash memory .. فتبدأ ت interact مع ال bootloader at runtime .. طب كويس قوي .

---

طب بشمهندس أنا كدا فهمت الوظائف بتاعت ال bootloader .. تعالى بقا ناخد ايه ؟ .. أنواع phases من ال bootloader .. يعني جوا ال bootloader موجود كذا كذا phase .. هل الأنواع دي موجودة في كل microcontroller ؟ لأ كل microcontroller ليه الشكل بتاعه .. أنا بديك كل حاجه وعابذك تبقا مخك كبير وفاهم كل حاجه .

---

## phases boot sequence inside Bootloaders 3

### Phase 1 - ROM Code

فيه بعض ال microcontrollers ببيجي معاها ال ROM Code أو ال BootROM ايه ده ؟ .. بصوا يا جماعه فيه بعض ال microcontrollers تقولك ايه ؟ .. أنا شركة NXP الشركة دي بتعمل ايه ؟ .. دي شركة Semi-conductor يعني هي اللي بتصنع ال microcontrollers تقولك أنا بعد ما صنعت ال microcontroller بتاعي حطيت في ال Flash memory وال

Flash دي أنا سميتها ال **ROM Code** مساحتها صغيرة خالص .. خلي بالك بقا خلي دماغك كبيرة .. ما تقوليش اسمها ايه يا بشمهندس وأنا ما لقيتش في ال SPECS ال ROM Code .. لأ الأسامي بتتغير من microcontroller للتاني .. المهم هي memory صغيرة خالص أنا كشركة بصنع ال microcontroller بتاعي روحت حطيت فيها كود جوا ال memory دي وال Non erasable memory يعني ما بتتمسحش .

---

يعني أنا حطيت memory صغيرة جوا ال microcontroller بتاعي وال memory دي الكود اللي عليها يتحط مرة ما يتمسحش وال reset section بتاع الكود بتاعي ده هو اللي عند ال entry point بتاعت ال processor .. طب يا عم ال vendor انت ليه عملت الحركة دي ؟ .. ما كنت تسييني أنا أخط الكود بتاعي في المكان اللي يعجبني عند ال entry point دي .. يقولك لا لا لو سمحت فيه حاجات أنا بعملها initialize جوا ال core نفسه لازم أنا اللي أعملها الأول .. خلي بالك مين اللي بيحط الكود ده ؟ ال manufacturer نفسه وحاطط الكود ما بيتمسحش يبدأ بعد كذا يقولك أنا بعد ما initialize حاجات مهمة منها ال SRAM .. مانت عندك من أنواع ال RAM ات فيه هناخدنا بعدين يا جماعه فيه نوع اسمه ال SRAM وال DRAM والحاجات دي .. أنا عملت initialize لل SRAM Controller اللي واصل بال SRAM Memory عشان أعرف أقرأ منها حاجه .. دي ميموري بس بعملها initialize لل control عشان أعرف أشوفها .. بعد ما عملتلها initialize روحت قريب software جواها اسمها SPL وفيه بعض الشركات بتسميه حاجات تانية زي ال init ROM .

---

المهم هو راح ل software تاني .. هو ال software اللي محطوط في ال ROM Code ده ال functionality بتاعته ايه ؟ إن هو عمل initialize لل SRAM وبعد كذا راح عمل jump على software في ال SRAM اللي هو ال SPL .. تمام طب هو كذا بيقا نوعه ايه ال software ده ؟ .. يقولك ده نوعه من أنواع ال bootloader .. ليه يا بشمهندس هو ملودش حاجه ! .. ايوه هو ملودش حاجه بس هو استخدم ليه ؟ عشان ي initialize راحه عشان عليها software يقومه فبيحطوه في ال references واحد من أنواع ال bootloader .

---

يقولك مش بس كذا .. هو أصلاً مين اللي قالني إن ال software اللي اسمها SPL ده موجود في ال RAM ؟ ال RAM دي هي volatile memory .. هو ليه أصلاً حطها في ال RAM ؟ هو جابها منين ؟ يقولك ممكن يكون فيه Flash memory تانية اسمها ال Code Flash مثلاً أو اسمها Flash Keroles .. انسى بقا الأسامي انت خلي دماغك أكبر من كذا .. ال Flash memory دي انت كنت حارق عليها ال software بتاع ال SPL ده وال ROM Code بتاع ال manufacturer راح قراها وخذ ال SPL ده حطه فين ؟ في ال RAM وبعد ما حطه في ال RAM شغله .

---

بشمهندس هو كذا معناه إن أنا ممكن أخط ال software بتاعي بدل ال SPL ده ؟ .. يعني أنا عملت keroles software ينفع أقول لل ROM Code ده ياخد Keroles software ويروح يحطه في ال RAM ؟ .. لأ خلي بالك بقا إنت الأول المكان اللي في ال RAM ده بتبقا RAM مكانها صغير بيحطوا فيها software صغير لأنها static RAM ودي غير ال dynamic RAM .. ال SRAM هينجي نعرفها انها غالية قوي فيستخدموها بس عشان السرعة وحنة حاجات وبتبقا مساحتها أصغر من ال DRAM .

---

اللي أنا عايز أقولهلك إنك ممكن تلاقي في بعض ال System on-chip ال DRAM بتبقا 4 KB بس وانت كذا على ال Stage دي ال ROM Code بالنسبالك هو عبارة عن bootloader .. ليه ؟ لأن هو نقل ال SPL من Flash حطه في ال RAM وبدأ يشغله من ال RAM .. طب ال bootloader ده ينفع أمسه ؟ لأ .. لأن اللي حطه هو ال vendor وأكد هو محطوط على ال entry point ؟ .. طبعاً ماهو لو مكنت محطوط على ال entry point بتاعت ال processor مكنتش قام أصلاً .

---

ال ROM Code ده يقولك هو عنده ال capability ان هو ي load small cunck of code ويحطه في ال RAM ، اللي أنا عايز أقولها لك ان ال ROM Code ده لو هو ال RAM صغيرة ممكن يروح يجيب ال SPL ده ويحطه في ال RAM أو فيه بعض ال ROM Code ما تحتاجش SPL .. يعني بتروح تقرا من ال Flash memory ال bootloader الأساسي وتروح تحطه في ال RAM .. يعني على حسب .

---

عشان ما تتلخبطش خيلنا بس نتكلم ان ال ROM Code ده bootloader راح لود ال SPL من ال Flash حطها في ال SRAM وال SRAM مساحتها صغيرة وال software اللي لوده ده اسمه ال SPL أو secondary program loader وده مساحته صغيرة جداً .. طب ده وظيفته ايه ؟ ان ده هيروح ي configure ال SD Card Controller .. يعني ده بعد كدا هيروح يشغل ال SD Card Controller عشان يطلع منها ال U-Boot وال U-Boot ده هو another bootloader يلوده من ال SD Card يروح يحطه في ال RAM بس المرادي وعايزك تركز بقا هو حطه في حاجه اسمها ال DRAM ودي أكبر بكثير جداً من ال SRAM .

---

ال SPL هو فيه كود ان هو يقدر بيرمج ال SD Card عشان ياخد another bootloader من ال SD Card يروح يحطها في ال RAM .. طب ليه يا بشمهندس الحركة دي ؟ .. بعد كدا ال bootloader بقا اللي اتحط في ال RAM اللي اسمه ال U-BOOT بدأ يقوم فيبدأ ي configure ال UART Controller عشان يعرضك ال messages دي .. فلما نيجي نفتح البوردة هنا نلاقي ان فيه حاجه اسمها ال Secondary Program Loader .. SPL وقالك Trying to boot from MMC1 ال MMC يعني ال SD Card .. بيقا ال SPL ده راح بيرمج ال MMC .

---

ولما راح بيرمج ال MMC راح خد منها ايه حضرتك ؟ .. راح خد منها ال U-BOOT اللي هو another bootloader ومساحته كبيرة بقا وخده راح حطه في ال DRAM ( dynamic RAM ) اللي مساحتها كبيرة فيعد كدا ال U-BOOT قام وطبعك messages بقا وبيقولك مستعد أخذ منك أوامر .. ايه ده يعني كدا يا بشمهندس احنا عملنا كام حاجه ؟ عندنا ال BootROM راح لل SPL عشان ال SRAM مساحتها صغيرة وده راح جاب ال U-BOOT من ال SD Card وراح حطه في ال DRAM بعد كدا راح خلى ال processor يقومه .. راح ال U-BOOT قام قالك بقا اكتبلي ال commands اللي انت عايزه .

---

بعد ما تكتب ال commands تبدأ انت تقوله بقا هاتلي يعم ال software بتاعي أو هاتلي ال Linux Kernel من ال Ethernet أو SD Card ثاني أو Flash Memory برا من أي حته انت عايزها .. وحطهالي في ال DRAM لأن ال DRAM كبيرة وابدأ شغلها .

---

بيقا كدا ال sequence ايه اللي حصل ؟ .. أول ما فتحت ال power ال processor راح على ال entry point لقا فيها كود اسمه ال BootROM ال BootROM ده مش أنا اللي كاتبه وما أقدرش أمسحه دا ال vendor حطهولي .. ال BootROM ده راح قرا ال jumpers ومن خلالها راح لل SPL خده وقومه ال SPL لما قام هو أكبر من ال BootROM وعنده Capabilities أكثر راح ي configure ال SD Card Controller مرة واحدة ولما عمل configure ليه راح قرأ ال U-BOOT وحطه في ال Dynamic RAM .. طب انت ليه يعم ال BootROM مروحتش لل DRAM من الأول ؟ ليه روجت لل SRAM ؟ يقولك أصل أنا ال BootROM ال vendor كاتبني أنا عندي Capability صغيرة قوي وال size بتاعي قليل ومكتوب بس ان أنا أخري أخذ ال software أحطه في ال SRAM .. فعشان كدا قولنا خلاص ال Software ده لازم يكون ال SPL لأن مساحته صغيرة .. هو ده اللي يتحط في ال SRAM عن طريقك وهو ده بقا عنده كود يقدر بيرمج ال DRAM فهو فعلاً قدر بيرمج ال DRAM فعرف ينقل ال U-BOOT من ال SD Card يحطه في ال DRAM بعد كدا ال U-BOOT قام .

---

ال DRAM واسعة .. فال U-BOOT قام بقا وقالك بس أنا هجيبلك أي حاجه وأحطها في أي حتة .. عايز ايه بقا شببك لبيك ؟ .. عايز أجيبلك من ال Flash ؟ ولا أجيبلك من ال QSPI برا ال Microcontroller أصلاً ولا أجيبلك من ال Ethernet عن طريق FTP Server ولا أجيبلك منين ؟ .. ما ال U-BOOT بقا عنده أكواد كبيرة بقا .. انتوا عارفين يا جماعة إن ال U-BOOT ده هو في الآخر Bootloader .. الكود بتاعه موجود على git و open-source لو فتحته هتلاقي فيه startup وفايلات C كتير .. يبدأ بعد كدا بقا هديله الأوامر إن هو يروح يجيبلي ال Software بتاعي اللي في الحالة هنا ال Software بتاعي هو ال Operating system نفسه .. يعني أنا ممكن من المرحلة دي يا جماعة أخليه يروح يشغل Keroles Software لو هو دا ال Bare-metal بتاعك اللي هيروح مثلاً ي configure ال UART أو ينور LED براحتك بقا .. بس طبعاً يعني انت عيب عليك يعني تستخدم بوردة فيها كل ال peripherals دي وجواها BootROM وراحت قومت SPL وراحت قومت ال U-BOOT عشان تروح تقوم Software انت كاتبه بي toggle LED !! بيقا عيب عليك ولا لأ ؟ كإنك جايب طيارة عشان تنقل الشقة اللي جنبك .

---

فيقولك لأ بقا حاجه بالحجم ده خليه تروح تقوملي OS فيدل ما تروح تقوم سوفت وير عادي غلبان انت تروح تقوم OS .. بس لا يمنع إن انت تروح تقوم ال software العادي الغلبان انت ممكن تعمل اللي انت عايزه .. هيروح بقا ال Kernel بتاعتك اللي هي ال Operating system بتاعك يجيبه من أي مكان يحطه في ال DRAM ويبدأ ال Kernel يشتغل وبعد كدا يخلي ال processor يشغله .. لما يشتغل بعد كدا يعمل ايه ؟ ال Operating system يقوم وبعد كدا ي initialize كل حاجه في البوردة بعد كدا يروح لل init process ودي موجودة في حاجه اسمها File System هنعرفها في ال Embedded Linux بعدين وبعد كدا يروح يقوم ال application .

---

ايه ده يا بشمهندس دا فيه application أهو ؟ خلي بالك فاكّر المحاضرة اللي فانت قولنا حاجتين .. ايه الفرق ما بين ال OS Application وما بين ال Bare-metal application .. ال OS Application هو اللي قايم فوق ال Operating System هو ده .. مانت قومت فوق ال Linux .. أما ال Bare-metal application هو اللي المفروض ال U-BOOT بعد ما يخلص يقومه بدل ما يقوم ال OS .. تمام تمام .

---

يعني كدا يا بشمهندس نقدر نقول ال concept اللي جاي ده ؟ .. إن ال BootROM ده بالنسبالي هو Primary Program Loader ، ليه ؟ لأن من غيره أنا مش هعرف أقوم حاجه .. يعني لو نفترض إن ال Flash memory الصغيرة اللي ال semi-conductor حطت عليها ال software بتاعها اللي ما بيتمسحش باظت فالجهاز مع السلامة .. لازم تغير ال chip .

---

فيه في ال references موجود terminology 1 بتسمي بطريقة معينة و terminology 2 بتسمي بطريقة ثانية فانت لازم تبقا عارف ال two terminology definitions عشان لو سمعتها حد بيتكلم فيها تبقا انت فاهم .

---

ال terminology 1 بتسمي ال bootROM ده إن هو Primary Program Loader .. ال terminology 2 ما بتسميش حاجه على ال bootROM بتسميه bootROM زي ما هو يعني .

---

بعد كدا ال bootROM ده راح قوم SPL .. يا جماعة SPL ده نوع من أنواع ال Secondary Program Loader ممكن تلاقي شركة طلعت Keroles PL يعني الاسم ده متأخدهوش .. ده موجود في الحقيقة وليه source code على النت بس ده

بي act هنا ك secondary program loader ليه secondary ؟ لأن ده intermediate loader هو loader صغير كل وظيفته في الحياة إنه لما يقوم يقوم ال UBOOT .. و 1 terminology دي مش مستخدمة قوي .

---

ال terminology 2 واللي هي مشهورة أكثر بسموا ال SPL ده ال 1st Stage Bootloader بعد كدا بقا راح قوم ال UBOOT اللي هو ال 2nd Stage Bootloader .

---

فتح مخك معايا ومتاخدش الكلام مسلم بيه .. يعني خلي دماغك ذكية .. هو أنا ليه احتاجت ال SPL هنا ؟ فكر كدا .. أصل انت هتشتغل في شركة فلازم تفكر .. أنا احتجت ال SPL هنا عشان كان ال SRAM مساحتها صغيرة ما ينفعش أخط ال UBOOT فيها .. ركز كدا .. ال UBOOT مساحته كبيرة وال SRAM اللي كانت في الميكروكنترولر ده كانت صغيرة فكان أخري ال ROM Code ما ينفعش أخط ال UBOOT في ال SRAM فاحتاج إنه يحطلي Secondary Program Loader يكون مساحته صغيرة يتحط هو في ال SRAM وهو بقا اللي يخلي ال UBOOT اللي يتحط في ال DRAM .

---

لو أنا كنت في micro-controller ال SRAM مساحتها كبيرة .. طب ما كنت من الأول ممكن تخلي ال ROM Code ينقلك ال UBOOT من ال SD Card يحطهولك في ال SRAM عطلول .. يعني ليه ما حطتش ال UBOOT في ال SRAM ؟ .. قولتلي عشان ال SRAM مساحتها صغيرة .. طب لو ال SRAM مساحتها كبيرة هل أنا محتاج ال SPL ؟ مش هتبقا محتاجه خلاص .. فخلي بالك على حسب نوع ال micro-controller انت بتحط ال sequence اللي يناسبك .. فعشان تفهم الحنة دي تعالى نشوف أمثلة .

---

## Booting Sequence Examples

### (Bootloaders on BIOS-based x86 (1

قبل ما ندخل على ال ايمبدد تعالى نشوف الكمبيوتر بتاعنا .. الكمبيوتر بتاعنا أول ما بتفتح ال power بيروح لل BIOS ال BIOS ده هو From ROM .. خلي بالك بقا إن ال BIOS ده هو ال BootROM .. ال BootROM في الحالة بتاعتنا هو ال BIOS وده software ، نوع من أنواع ال bootloader اسمه BIOS وهو ده اللي بي act as a BootROM وده يا جماعه ما بيتمسحش .. ده بيتشال .. يعني لو ال BIOS بعد الشر باظ عندكوا .. لازم تغير chip ال BIOS يعني ايه ؟ يعني ال ROM IC اللي عليها ال BIOS Software اللي بيتشغل ك primary program loader اللي من غيره مش هتتعرف تشغل الجهاز .. بعد كدا ال BIOS بيروح ل stage 1 من ال raw storage يعني ال hard disk يعني فيه في ال C partition بعض الفايلات بت act as a stage 1 bootloader ودي بعد كدا بتقوم فايل ثاني بي act as stage 2 وبعد كدا ال stage 2 يروح يقومك ال Kernel ويحطها في ال RAM ويبدأ بقا في ال DDR RAM فيبدأ بعد كدا يشغل ال Linux Kernel بتاعتك على الكمبيوتر .

---

ال software المحطوط على ال BIOS المشهور قوي اسمه grub فممكن تسررش عليه وتشوف ال software بتاعه .

---

### Booting on ARM Microchip AT91

تعالى نبص إيه اللي بيحصل فيها .. من ال Specs بيقلك إيه ؟ .. أول ما ال power يقوم ال ROM Code اللي حطه ال vendor اللي هي Microchip حطاه هو ما ينفعش يتمسح وهو محطوط في ال ROM وهو ده اللي هيروح يقوم حاجه اسمها ال AT91BootStrap يعني هو عمله Loading من ال sd card وحطه في ال SRAM وبعد كذا ال software اللي في ال SRAM ده عمل إيه ؟ راح جاب ال UBOOT من ال sd card راح حطه في ال DRAM وبعد كذا ال UBOOT لما قام راح قوم ال Linux من ال SD Card راح حطها في ال DRAM وشغلها .

---

إيه ده يا بشمهندس .. يعني نقدر نقول كذا إن ال ROM Code اللي الشركة حطاه راحت قومت ال AT91BootStrap وده نقدر نقول عليه 1st Stage Bootloader من 2 terminology وده لما قام راح قوم ال UBOOT اللي هو بيقوم هناك 2nd Stage Bootloader .. شوفت بقا الدنيا ماشية ازاي .

---

## Booting on ARM TI OMAP2+ / AM33xx

البوردة دي أول ما قامت راحت لل ROM Code بتاعت ال Vendor .. ال ROM Code هو اللي عنده ال Entry point بتاعت ال processor راح قوم ال SPL اللي راحت قومت ال UBOOT اللي راح قوم ال Kernel .

---

## Booting on Marvell SOCs

دي عندها ال power يقوم .. ال ROM Code يقوم يقوم ال UBOOT .. طب هنا يا بشمهندس ال UBOOT موجود في ال SDRAM .. ببقا دا معناه إن البوردة دي ال SRAM بتاعها كبير بقدر يشيل ال UBOOT مرة واحدة عشان كذا ما احتاجناش هنا نخط ال SPL .. هي على حسب احتياجك في ال microcontroller بتاعك .. فانت فتح دماغك بقا .. خلاص بقا .. انسى ان انت تقول دي كذا لازم ألقبها كذا .. لا أنت فاهم وعلى حسب اللي موجود وبتعرف تقرا وبتعرف تحكم .

---

## Zynq-7020 soC from Xilinx

هنا ال BootROM قام راح قوم ال 1st stage bootloader اللي هو FSBL وده قومت ال UBOOT وده قوم ال Linux .

---

طب بشمهندس أنا كذا فهمت إيه بقا ؟ .. فهمت ان فيه حاجتين .. فيه بعض البوردة ملهاش BootROM أصلاً والله البوردة جايالك بيقولي ال entry point عند ال address الفلاني فانت بتخط ال software بتاعك عند ال entry point ده وتشتغل ومغيش bootROM أصلاً .

---

فيه بوردة تانية فيها BootROM وال BootROM هو اللي بيقولك أنا بعد ما بقوم بروح ل address الفلاني فساعتها انت بتخط ال software بتاعك عند ال address الفلاني فكأن ال software بتاعك هو اللي مكان ال FSBL ده .. أصل انت مش هتقوم حاجه كبيرة .. انت حطيت ال Bare-metal بتاعك مكان ال FSBL ده .

---

يا إما انت تظبط ال ROM Code لما يقوم يروح يقومك انت بدل ال SPL .. يعني انت ما تحطش SPL أصلاً .. ما تحطش ال UBOOT انت تحلي ان ال ROM Code يقومك انت علطول .. تقدر عادي بس أصعب .. ليه ؟ لأنك هتحاول ت configure ال ROM Code .

يا إما تقوله روح لل SPL وروح لل UBOOT وأنا ال terminal بتاعت ال UBOOT هديله أمر هقوله روح جيبلي من على ال SD Card بتاعتي ال software اللي اسمه كزا شيلهولي حطهولي في ال RAM وشغله .. بس يا معلم خلصت على كذا .

يا إما أصلاً ما يكونش فيه ROM Code ومغيش حاجه من الكلام ده وعرفت ال entry point من ال processor حطيت ال software بتاعي وخلص كذا .

---

فانت خلي دماغك كبيرة .. فيه Flavors كتير .. انت على حسب البوردة بتاعتك تتعامل معاها وتبقا ذكي .. يعني أنا كنت مع بروجكت قبل كذا مع شركة قولتلها انتي مش هينفع .. كانت عايزه من ال ROM Code تطلع من ال UBOOT علطول على ال Linux قولتلها مهما حصل مش هتعرفي .. قالولي ليه ؟ .. قولتلهم بسيطة خالص .. ال SPECS بتقول ان الشركة اللي صنعت البوردة اللي حضراتكم مستخدمينها دي عليها ROM Code وال ROM Code ده بيروح يقرأ فايل من ال SD Card اسمه مثلاً أي حاجه والفايل ده بيروح يحطه في ال SRAM .. وأنا روحت عملت check لل SRAM من ال SPECS لقيتها 12 كيلو بايت وروحت عملت check لل UBOOT لقيتها 24 كيلو بايت .. بيقا حضرتك استحالة ت run ال UBOOT بعد ال BootROM .. قالولي ليه ؟ .. قولتلهم make sense ازاي عايز تحط 24 كيلو بايت في ال SRAM مساحتها 12 كيلو بايت .. حضرتك ما ينفعش .. مقدمناش إلا حلين .. يا إما حضرتك تروح تتفق مع الشركة اللي صنعت البوردة إنها تغير ال ROM Code ده إنه ما يروحش لل SRAM يحط حاجته لأن ال SRAM دي 12 كيلو بايت وانت عايز تحط UBOOT ب 24 كيلو بايت .. يا إما بقا تاخد ال SPL مساحته 3 كيلو بايت تحطه في ال RAM وده يروح ياخد ال UBOOT ويحطهاك في ال DRAM اللي مساحتها كبيرة .. قالولي خلاص انت متأكد من الكلام ده ؟ قالولي أه .. دا بعد ال investigation اللي أنا عملتها بعد أسبوع بقولكم إن ده اللي أنا وصلته .. راحوا دوروا وردوا علينا قالولنا Thank you فعلاً اللي انتوا قولتوه صح .. قولولنا بقا نجيب ال SPL ده منين ونعمله compile ازاي عشان نحطه في ال SRAM عشان يقوملنا ال UBOOT .

---

الموضوع بسيط .. ال embedded يا جماعه على قد ما هتحبه وتفهمه على قد ما هيبقا سهل بالنسبالك لأنك بتلعب Logic .. المهم ما تخافش منه .. متاخدش الحاجه ايه ده أنا معرفتهاش وتخاف .. لأ انت خليك فاهم وفكر .. لو اتسألت سؤال انت مش عارفه فكر فيه وحاول باللي انت فاهمه ك concept صح تطلع الإجابة .. يعني وأنا بتكلم معاهم أنا كنت مرعوب ، ليه ؟ لأن أنا بكلم ناس بيشككوا في اللي احنا بنقوله وأنا شايف قدامي مهندسين كتير بس أنا متأكد لأن أنا قرئت كتير فأنا متأكد من ال SPECS اللي قريتها .. فأنا بيني وبينهم ايه ؟ ال SPECS .. فده اللي أنا عايز أقولهمكم .. ما تخافش من حاجه .

---

احنا كذا خلصنا أنواع ال Bootloader ودلوقتي بقا نبدأ في محاضرة النهاردة .

## This lecture is very important

محاضرة النهاردة دي مهمة جداً لأنك لو فهمتها صح هتبقا محترف embedded .. يعني ايه محترف embedded ؟ يعني هتخدم على كل اللي جاي .. ولو كنت في انترفيو هتس بتقل وانت بتتكلم كذا الناس هتس إن انت تقيل .. تقيل في المعلومات يعني .. فالمحاضرة الجاية دي فيه نوعين من ال engineers هيطلعوا منها .. يا إما هتطلع منها حابب ال embedded قوي وبدأت تتمكن منه .. يا إما هتطلع منها عايز تحول أدبي ومش عايز تكمل الدبلومة ولا تكمل ال embedded خالص .. من النقطة دي انت اللي هتقرر .



احنا في المحاضرة دي هنبدأ نتعلم أسلوب جديد .. ايه هو الأسلوب الجديد ده ؟ احنا هنخش في journey هنعدي فيها على ان احنا نتعلم من خلال ال Lab .. يعني احنا هناخد Lab ونقعد ونقعد نمسك كل حنة فيه نبدأ نتعلمها .. بدل ما نقعد ناخذ نظري بس .

---

طب احنا هنتعلم ايه ؟ تعالى نفكر كذا .. احنا هنعمل Lab1 وده اللي هنعمله النهاردة .. Lab1 ده هنكتب فيه Bare-metal application أو Bare-metal software .. يعني ايه Baremetal Software ؟ .. يعني مفيش Operating System .. أنا هيقا كاتبه according لل Specs وهروح أعمل run ليه على ال ARM VersatilePB ودي بوردة شركة ARM عاملها .

---

طب ايه اللي أنا هطلع بيه من اللاب ده ؟ .. أول حاجة أنا عايز أكتب application بيقا عايز أعمل ايه ؟ أكتب C Code files وعايز أعمل Linker Script وعايز أكتب startup وعايز أعرف أستخدم Cross toolchain ايه وأكتب Makefile يعمللي automation لل build .

---

بشمهندس ، طب ايه هو ال Cross toolchain ؟ ده خدناه المحاضرة اللي فاتت .. احنا في ال Lab ده عايزين نشغل from scratch مش هنعتمد على أي IDE .. احنا هن create بايدينا Bare-metal application from scrach من غير أي اعتماد على أي حاجة .

---

هنكتب application .. ال application ده وظيفته ايه في الحياة ؟ ده وظيفته إنه يخلي ال processor يروح ي configure ال UART والبوردة دي فيها UART 4 احنا هنخليه يكلم UART0 .. بيقا ال Chip دي لو دخلنا فيها هنلاقي فيه processor وفيه Flash و RAM وفيه Peripherals كتير .. احنا المثال ده هنخليه Very Simple .. احنا هنتعامل كإن عندنا memory واحدة بس هنشغل عليها كل حاجة .. بشمهندس يعني دي بالنسبالنا هتبقا هي ال RAM ؟ اه هي دي هتبقا ال RAM بالنسبالنا .

---

بشمهندس احنا مش هننقل الحاجات بقا وننقل ال data section من ال Flash لل RAM ؟ الحنة دي هنخليها لل Lab الجاي .. خلوا بالكم اللاب ده هنتعلم منه الأساسيات .. اللاب اللي جاي هيبقا أصعب هيبقا فيه كل حاجة بقا .. هتبقا انت اتعلمت الأساسيات فهنخش بقا على ال Advanced بقا .

---

بيقا اللاب ده أنا عايز أكتب Software يخلي ال processor ال ARM926EJ-S ده يروح لل Bus ي configure ال UART0 عشان تبعت على ال pins بتاعتها Characters byte by byte تخليه يطبع جملة Learn-in-depth: your name .. طب احنا نعرف نعمل الكلام ده واحنا لسا ما اتعلمناش ايمبدد ؟ هو ال Embedded عبارة عن ايه ؟ Embedded C واحنا دلوقتي بنتعلم Embedded C .. بعد كذا بنقرا من ال Specs يعني ايه UART والكلام ده هنتعلمه في الكورس اللي جاي .

---

فأنا حنة ازاي نكلم ال UART والكلام ده أنا هسهلك بس حنة ازاي نكتب Baremetal Embedded C ل application هو ده اللي أنا عايز أطلع بيه .. طب يا بشمهندس نعملها ازاي ؟ احنا نعمل application اسمه app.c وهنعمل فايل اسمه Uart.c وفايل اسمه Uart.h .. ال Uart.h ده فيه ال header بتاعت ال driver اللي اسمه Uart.c وهيصله including من ال Uart.c و including من ال app.c .



---

هنعمل فايل ثاني احنا اللي هنكريته بإيدينا وهو ال startup.s اللي هو ال assembly code لل startup وهنعمل فايل ثاني اسمه ال Linker Script .. والمفروض بقا نتعلم ان احنا ندخل ال UART وال app.c على ال compiler gcc والكومبيلر يطلعنا assembly تخش على ال assembler اللي اسمه as يطلعلي Uart.o و App.o .. يعني ال object file لل UART وال object file لل application .

---

طب يا بشمهندس انت ليه ال startup مدخلش على ال compiler ؟ .. عيب عليك بقا ال startup ده assembly فهو مش محتاج compiler ي translate ال assembly ل assembly .. ما هو assembly أصلاً ! .. فهو هيدخل على ال assembler علطول ويطلع منه startup.o .

---

بيجي ال Linker اللي جواه حاجه اسمها Linker/Locator يعني ال Locator دي جوا ال Linker .. ال Linker من اسمه كدا يبدأ ي link ال startup.o مع ال uart.o مع ال app.o وباستخدام ال locator يقرأ حاجه اسمها ال Linker script اللي يقول كل سكشن يتحط عند address كام ويطلعلي حاجه اسمها ال executable file اللي هو اسمه Learn-in-depth.elf .. ال executable file فيه debug information وفيه حاجات كتير قوي .. أنا دخلته على toolchain بتاعتي .. اللي هي ARM-Cross toolchain عشان أستخدم Binary Utility اسمها strip .. عشان أ copy منه الجزء اللي مفيهوش debug information الجزء ال binary بس وأطلعاه في شكل hex يعني Learn-in-depth.hex وال hex دي هحطها في ال RAM ولما أحطها في ال RAM ابدأ أشتغل .

---

خلي بالك يا عبدالله هنا قال الفرق ما بين ال .hex وال .elf .أهو .. إن ال .elf بيبقا فيه Debug information فده لو انت عايز تعمل debug هتستخدمه .. لكن لو انت عايز ت run بس استخدم ال .hex.

---

طبعاً ال processor ده من ال SPECS ليه entry point عند address 0x10000 يعني أنا من غير ما أتكلم البوردة دي مفبهاش Bootloader .. قال entry عند 0x10000 معناه ايه ؟ ده معناه إن أنا أحط ال reset section من جوا ال startup بتاعي عند 0x10000 .. طب دي هعرف أعملها يا بشمهندس ؟ ما ده اللي هنتعلمه .. بعد كدا ال section بتاعي ده يبدأ ينده على ال main ويعمل initialization ويشغل ويحدد ال stack عند ال address ده ويحطه size ب 1000 بايت .. فال ال Stack\_top بيتحط عند بداية ال stack .

---

بشمهندس أنا أعرف أعمل الكلام ده ؟ اه .. طب احنا المحاضرة اللي فاتت خدنا ايه من الكلام ده كله ؟ خدنا ال ARM Cross Toolchain يعني المحاضرة اللي فاتت عرفنا يعني ايه Native Toolchain ويعني ايه Cross Toolchain .. المحاضرة اللي فاتت عرفنا يعني ايه Binary Utilities جوا ال ARM Cross Toolchain النهاردة هنستخدمها .

---

بشمهندس أنا ما أعرفش أكتب Linker أنا ما أعرفش يعني ايه Startup .. ما هو ده اللي هنتعلمه و هنتعلمه من خلال ال Lab .

---

أنا رافعلك ال installation بتاع QEMU وبتاع GNU ARM Embedded Toolchain وحاططهملك في لينك دراييف في المحاضرة .. أول واحد ده QEMU ده بيعمل ايه ؟ ده اللي بيخليك ت simulate البوردة .. طب وال GNU ARM ؟ ده ال Cross Toolchain اللي هنستخدمه .. بعد ما تسطبهم هيطلع معاك 2 فولدر واحد لكل واحد فيهم .

---

لو فتحت دلوقتي الفولدر اللي انت مسطب فيه ARM GNU هتلاقي جواه ايه ؟ .. هتلاقي جوا الفولدر اللي اسمه bin كل الحاجات بتاعت ARM ال Cross Toolchain وال Binary Utilities .. في نفس الوقت هتلاقي فيه حاجه اسمها ال make ودي كانت مع ال MinGW اللي انت سبطته لما كنت بتستخدم ال git .

---

ايه هو QEMU بقا ؟ هو من اسمه هو اسمه Quick Emulator بيعمل ايه بقا ؟ عنده مجموعة من ال supported processors زي cortex-m3 أو cortex-a9 إلخ ... طب فين البوردرات اللي بي support ليها ؟ لو انت روحت للمكان اللي مسطب فيه QEMU ودخلت جواه كتبت ال Command ده qemu-system-arm.exe وقولته machine help- .. هيجيبلك كل البوردرات ال supported على qemu وهتلاقي من ضمنهم ال Raspi 2 وبرضو ال Versatile epb اللي هتشتغل بيها النهاردة وغيرهم كثير .

---

ال QEMU ده ميزته ايه ؟ ميزته إن ال Microcontroller انت شايف داخلها ايه .. يعني البوردة انت ما بتشوفش البوردة دي جواها ايه .. ليه ؟ لأن البوردة دي بتبقا في الآخر Black Box بالنسبالك ف qemu هو simulator بي simulate البوردة فتقدر من خلاله ت Debug ال Software بتاعك وتشوف البوردة جواها ايه أو بيحصلها ايه .

---

بيقا كدا ايه اللي انت تمتلكه عشان نبدأ نعمل ال binary بتاعنا ؟ .. خلوا بالكوا الرحلة بتاعنا احنا عايزين نكتب our own baremetal انت اللي تكتبه from scratch ما تخافش من حاجه .. فاحنا معانا ايه ؟ احنا معانا ال Cross Toolchain اللي هن compile بيها ومعانا البوردة .. يلا نكتب الكود .

---

أول حاجه يلا نكتب الكود .. طبعاً ال Embedded Very Easy لو انت حبيته .. طب احنا عايزين نكتب كام فايل ؟ 3 فايلات Uart.c , App.c, Uart.h طب ايه هدفنا ؟ .. أنا عايز لما أكتب ال 3 فايلات دول أعمل compilation ليهم وأطلع ال .o فايل .. يعني أنا عايز أوصل في الآخر من ال stage دي إن أنا أطلع ال Uart.o وال App.o عشان بعد كدا أعرف أعمل Link ليهم .

---

البوردة دي فيها UART 4 احنا هتشتغل على UART0 .. طب ايه هو ال UART0 يا بشمهندس ؟ هناخد في ال Serial communication course .. بس دلوقتي هو Serial interface بيعت Serial bytes .. يعني بيعت sequential ل byte by byte مع بعض .

---

طب يا بشمهندس أنا لسا ما اتعلمتش ازاي أفتح ال SPECS واقراً والكلام ده .. عشان انت لسا ما اتعلمتش الحتة دي أنا بقولك أنا فتحت ال SPECS بدالك في كورس ال Embedded C ولقيت إن ال UART0 ده موجود عند ال Address ده 0x101f1000 .. يعني ايه موجود عند ال Address ده يا بشمهندس مش فاهم ؟ يعني البوردة لما تيجي تفتحها ال processor واصل بال UART وال UART عنده Registers عايز تيرمجها عشان بيعت ويستقبل حاجه فال Address بتاع ال UART0 ده على ال Bus هو كان ال Address ده 0x101f1000 .

---

طب ال Register أوصله ازاي ؟ هتجمع ال offset بتاع ال Register مع ال Address ده فتقدر توصل لل Register نفسه .

---

طب قولني بقا Register ايه اللي محتاجه ؟ .. من ال datasheet لقينا إن البوردة دي عندها ال UART بتاعها جميل جداً ومن أسهل ال UART في الدنيا اللي هتتعرض ليه .. ما أعتقدش هتلاقي أسهل من كذا .. عنده Register اسمه UARTDR يقولك ده اللي بيكتب على الحاجه اللي تتبعت .. ده For words to be transmitted يعني ده لما بتكتب عليه بيعت .. بيقولك لو ال FIFO Enabled فالداتا اللي اتكتبت عليه هتتبع طول وتروح لل Transmit Side

---

طب ال Register ده موجود عند Address كام ؟ .. أنا عايز أعرف ال offset عشان أجمعه على ال Base Address عشان أعرف أوصله .. هو عند 0x0 .

---

هنتفتح دلوقتي الفايلات بقا App.c, Uart.c, Uart.h ونبدأ نكتب ( أنا غالباً مش هكتب اللي هيحصل هنا لأن ده كود عادي مش حاجه ينفع تبقا Concept علم في ال Embedded يعني ) .

---

هروح في فولدر فاضي وأفتح ال Bash بتاعي بيكون أسهل في التعامل يعني .. هقوله :

touch uart.c uart.h app.c

هتلاقيه عمل creation ل 3 فايلات دول .

---

تعالى نشرح عملية ال define أو ال Macros .. انت عندك ده :

```
((define UART0DR ((volatile unsigned int*)((unsigned int*)0x101f1000#
```

ده معناه ايه ؟

ده معناه ال pointer اللي هو volatile unsigned int\* اللي بيشاور على ال Address ده اللي هو 0x101f1000 اللي احنا عاملين casting ليه بال type ده unsigned int\* عشان يكون Address بدل integer .. وعملنا \* من برا يعني حط جواه ال Value اللي انت هتخطها .

---

انتوا عارفين إن ال Macro بيحصله Text Replacement في ال Pre-compile فهيشيل الاسم اللي هو UART0DR ده ويحط السطر الطويل اللي ع اليمين ده .. فالشكل ده بيقول ايه ؟ ال pointer اللي بيشاور على ال Address فلان الفلاني حط جواه ال Value كذا .

---

ناقص حاجة بقا .. أنا عايز ال pointer ده بشار على ال Address بتاعه ده علطول ما يتعملوش increment .. هو كدا كدا مش هيتعمله increment لأن مفيش variable يتعمله increment فبراحتك انت لو عايز تحط const وتقول إن ال pointer is cont ماشي ولو مش عايز مش هتفرق .. وده الشكل بتاع لما نحط const :

```
((define UART0DR ((volatile unsigned int const)((unsigned int*)0x101f1000#
```

---

فأنا كدا عرّفت ال Register اللي هستخدمه .. بعد كدا بقا تعالى نبدأ نفكر مع بعض احنا عايزين ايه ؟ .. احنا عايزين function نسميها Uart\_Send\_String لأن احنا عايزين نبعت String وك parameters عشان أستقبل string فيها بيقا هحط pointer على ال string ده فبيقا :

```
{ }( void Uart_Send_String (unsigned char* P_tx_string
```

بيقا ده أنا هبعت pointer ال pointer ده بشار على بداية ال string اللي هبعثها من ال main فأنا كدا استقبلتها في ال function دي .. بعد كدا بقا هعمل ايه ؟

---

ال string دي هتجيلي فيها characters لحد ال Null فأنا عايز أعمل polling على ال Null ولما ألاقي ال Null أقوله بس كفايه كدا فأنا هقوله \* على ال pointer ده كإني بخش جواه وأخليه طول ما هو مش بيساوي ال ١0 .. يعمل ايه بقا ؟ أخليه يكتب على ال UART0DR .. هنعمل بس casting الأول ب unsigned int ونكتب عليه ال value بتاعتك اللي انت بشار عليها دلوقتي اللي هي \*P\_tx\_string والكود كدا :

```
(void Uart_Send_String (unsigned char* P_tx_string
```

```
}
```

```
( 'while (*P_tx_string != '\0
```

```
}
```

```
; (UART0DR = (unsigned int) (*P_tx_string
```

```
{
```

```
{
```

---

بيقا هو دلوقتي أنا استقبلت ال string في ال function وشار على بداية ال string .. ببص على اللي جوا ال string وطول ما هو مش بيساوي Null حطلي ال value بتاعته جوا ال UART وعشان ال UART من نوع pointer to integer فأنا بحول ال unsigned character بتاعت ال string ل unsigned int عشان خاطر تتحط جوا ال UART .

---

بص يا عبدالله هو مش ال string ده عبارة عن مجموعة من ال characters ؟ اه فالحنا هنبعت ال string عادي لل function بتاعتنا character by character بس احنا لما نيجي نحطه جوا ال Register بتاع ال UART عشان بيعته لازم نعمله Casting بنفس ال data type بتاع ال UART اللي هو هنا unsigned int .. فدي فكرة ال casting في الحتة دي .

---

بعد كذا المفروض أعمل ايه ؟ .. هي increment بقا ال pointer ده عشان أدخل على ال next character ولما أوصل لل null هخرج من ال Loop دي :

++;P\_tx\_string

---

الكود هتلاقه في ال repo لو عايز تتابع معاه يا عبدالله يعني .

---

احنا كذا عملنا ال uart.c احنا محتاجين ايه بقا في ال uart.h ؟ أنا محتاج أعمل prototype لل function دي ومحتاج أعمل ال safety اللي هي إن أنا أحمي ال header بتاع ال UART بتاعي بال #ifndef .. شوفت الايمبدد جميل وحلو ازاى بس لو انت حابه .

---

هو احنا ليه عاملين Casting لل Address نفسه ؟ لو معملتهلهوش Casting مش هيحصل مشكلة بس هيطلعلك Warning يقولك ايه ؟ احنا عملنا مثال على Eclipse وشوفنا فيه ان احنا لما بنعرف pointer ببيساوي رقم .. هو الرقم ده Address بس هو بالنسبة لل Compiler مش عارف إن ده Address هو بيقول إن ده integer number مع إن ال integer number هو ل Address هنا .. فأنا عشان أمنع ال Warning ده هنا بقوله خلي بالك ال integer ده مش مجرد integer بس .. لأ ده integer ل Address أو بيمثل Address .. فيشيل ال Warning .. لكن لو انت معملتتش Cast فهو هيشغل عادي مفيش مشكلة لإن هو في الآخر Address .

---

الحته اللي جاية دي كتابة كود ال application بس لو فيه أي comments هكتبها .

---

تعالى بقا نكتب ال app.c أول حاجه هن include ال uart.h شيء طبيعي ، ليه ؟ عشان أعرف أستخدمها في ال main في ال application بتاعي .. تعالى نعرف بقا :

} (void main (void

{

يعني function ما بتاخدش حاجه وما بترجعش حاجه وسميتها ال main .

---

دي أول ما يخش جواها عايزه يعمل ايه ؟ أو يروح ينده على مين ؟ على ال Uart\_Send\_String ( ) فانكشن طب نباصيلها ايه بقا ؟ عايزين نباصي ليها string .. تعالى نعرف في ال global كدا :

“ unsigned char string\_buffer[100] = “ learn-in-depth : Abdallah

---

دي ال array of characters اللي هنبعتها على ال UART واللي هنباصيها لل function اللي هي Uart\_Send\_String والكلمة اللي جواها دا الكلام اللي أنا هبعته وخلي بالك أنا ببعث لل function دي pass by reference فهبعث ليها ال address اللي هو اسم ال array .

---

فاضل ايه بقا ؟ يلا نعمل compile مع بعض .. لما نيجي ن compile نعملها ازاي ؟

## Let us generate (app/uart).o object files

هفتح ال terminal بتاعتي وأنا مسطب في ال directory اللي برا ARM toolchain اللي أنا قولتكم عليها وجوا ال ARM ده فيه فولدر اسمه bin لو أنا عملت ليه ls هلاقي فيه كل حاجات ال compiler :

```
/ls ../ARM/bin $
```

---

أنا بقا عايز أشاور على ال Directory ده فيقولك عشان تشاور على المكان اللي فاتت ده بتعمل حركة اسمها :

```
export PATH $
```

ال PATH ده يا جماعه environment variable فيه كل ال paths اللي انت بتشاور عليها في ال windows بتاعك .. طب بشاور عليها ازاي حضرتك ؟ بتشاور عليها ان انت بتعرف لك environment variable في ال system ال paths بتاعت ال tools بتتعرف تحت ال path directory .

---

طب أنا عايز أعمل ايه ؟ عايز أعمل كدا :

```
export PATH=../ARM/bin/:$PATH $
```

خلي بالك ممكن يكون ال directory عندك في مكان ثاني عادي يعني .

أنا كدا عملت ايه ؟ بال command اللي فوق ده ؟ عشان لما أكتبه :

```
-arm
```

وأدوس tab يعمل كدا يملأها لوحده :

```
-arm-none-eabi
```

طب هو عرف منين ؟ مع ان ال directory اللي أنا واقف فيه ده مفيهوش arm-none-eabi-gcc !

---

عرف منين ؟ هقولك ما هو أنا كنت فاتح ال terminal في ال file بتاع الأكواد وعملت كدا إن ال path بتاع الأكواد واللي هو ببشاور عليه في ال terminal دي ي include معاه ال path اللي هو فيه ال binaries بتاعت ال toolchain في ال directory بتاعها .

---

فده كدا معناه ايه ؟ لو أنا قولتله :

`which arm-none-eabi-gcc.exe $`

هيجيبلك ال `directory` بتاعه ويقولك حضرتك أنا جاييه من المكان ده .. طب انت عرفت منين يا `terminal` إن أنا جاييه من هنا ؟ .. مانت حطيتھولي في ال `path` .

---

طب لو أنا عايز أعمل الموضوع ده عارف كنت هتقعد تعمل ايه ؟ كنت هتقعد تقوله كل شوية تكتب ال `path` اللي هو ده :

`ARM/bin/arm-none-eabi-gcc.exe/.. $`

هل أنا هقعد بقا كل مرة أكتبه ال `path` ؟ طب مانت تحط ال `directory` كله اللي انت مسطب فيه في ال `path` بتاعك اللي انت واقف عليه وتشتغل علطول يا معلم .. فاهمني ؟ أسهل ليك .

---

طب **ليه** `arm-none-eabi` ؟ خدناها المرة اللي فاتت .

---

لو قولنا :

`arm-none-eabi-gcc.exe --help $`

دي بيديني كل ال `options` اللي ممكن أديها لل `compiler` .

---

احنا عايزين نطلع ايه يا بشمهندس ؟ عايزين نطلع ال `object file` يعني عايز أ `compile and assemble` ولكن `do not link` يعني أنا مش عايز أعمل `link` طب ليه ؟ لأن أنا لسا معملتش ال `linker script` أنا لسا بتعلم أنا بعمل كل حاجة بإيدي `from scratch` أنا عايز أطلع ال `object file` بس بيقا أنا هستخدم ال `-c` .

---

أنا ممكن أكتبه كدا عشان أعمل `compilation` من غير `link` :

`arm-none-eabi-gcc.exe -c app.c -o app.o $`

ممكن أعمل كدا ؟ اه ممكن تعمل كدا بس فيه غلطة انت محددتش ال `architecture` أو ال `processor` اللي بعمله `compile` ما فيه كذا `core` في ال `ARM` فيه `ARM-Cortex-A9` وفيه `ARM-Cortex-A7` وكثير غيرهم .. انت محددتش نوع ال `processor` اللي تعمله `compile` فيقولك طب أعرفها منين ال `flag` اللي يعمل كدا يا بشمهندس ؟ هقولك خش على ده :

[https://gcc.gnu.org/onlinedocs/gcc-9.3.0/gcc/GCC Manual](https://gcc.gnu.org/onlinedocs/gcc-9.3.0/gcc/GCC%20Manual) :

---

ده هيدبك ال `options` اللي ال `gcc` بياخدھا عشان يحدد نوع ال `processor` فيه `option` يا جماعه اسمه `mcpu`- ايه ده يا بشمهندس ؟ .. ده يعني ال `machine` ال `CPU` بتاعها ايه .. ده انت بتديله اسم ال `processor` .. يعني يا بشمهندس أعمل ايه

؟ هزود على ال command اللي فوق ده اسمه mcpu -عشان أقوله اسم ال processor واسم ال processor بتاعنا هو arm926ej-s فهيبقا ال command عامل بالمنظر ده .

```
arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s app.c -o app.o $
```

---

طب يا بشمهندس أنا عايز أعمل include لل header اللي أنا واقف فيه .. أعمل كذا ازاى ؟ عشان تعمل كذا بتكتب -I- وتحط ال input directory اللي فيه ال header اللي انت بتعمل include ليه .. ولو أنا بعمل include لل header لنفس المكان اللي أنا واقف فيه فعمل -I- فهيبقا ال command عامل معايا بالمنظر ده :

```
arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I app.c -o app.o $
```

---

طب يا بشمهندس أنا عايز أطلع ال application بتاعي ب debug information .. لو عايز تحط debug information هتدوس -g .

---

طب بشمهندس أنا عايز أعمل optimization nothing أو أحدد ال level بتاع ال optimization .. هتعمل O1- مثلاً لو عايزه level 1 بس هو بيكون by default ب 0 يعني مفيش optimization .

---

فهيبقا ال command في الآخر عامل معانا بالمنظر ده :

```
arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . app.c -o app.o $
```

---

فكدا احنا طلعا ال app.o تعالى نعمل بقا نفس الكلام مع ال uart.c ونطلع بقا ال uart.o :

```
arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . uart.c -o uart.o $
```

---

خلي بالك بقا فيه اسم لل object file ده والاسم ده ببسألوا فيه في الانترفيوهات دلوقتي اسمه relocatable binary خلي بالك من الاسم ده لإن الاسم ده تحتية معنى عميق قوي هنفهمه دلوقتي .. ايه ال Relocatable binary ده ؟ يعني ال binary اللي لسا ما اتعملهوش **reallocation** أو **located** عند **address** معين .. لسا binary بس مش محطوط عند **Addresses** physical لسا هيتحط .

---

خلي بالك اللاب ده احنا لو عايزين نعمله ممكن نعمله في 10 دقائق بس الفكرة ان احنا بنتعلم من خلال اللاب ودي الطريقة اللي أنا بقولكم عليها .

---

**(Navigate the .obj files (relocatable images**



يعني ايه relocatable image ؟ يعني obj file .. طب يعني ايه obj file ؟ تعالى بقا نخش جوا ونعمل analysis لو عندنا obj file نفهمه ونعرف ايه اللي جواه .. بيقولك بقا ال obj file اللي انت طلعتة ده هو binary code منا عارف ان هو . binary code

---

جواه سكاشن ، لا السكاشن دي أنا مش عارفها الحقيقة .. لا ما احنا قولنا بقا إن ال instructions موجودة في ال text section وال global data موجودة في ال data section وال uninitialized global data موجودة في ال bss section .

---

لو عايزين نفهم ال binary بتاعنا ده هنستخدم binary utility حلوة قوي في ال ARM toolchain اسمها objdump .

---

## (Using ARM-Cross toolchain Bin Utilities (objdump

arm-none-eabi-objdump.exe \$

وعشان تعرف هي بتشغل ازاي هتقوله :

arm-none-eabi-objdump.exe --help \$

---

من ضمن الحاجات المهمة بالنسبالنا -h و -d :

ال -h ده يجيبك ال headers و -d ده يجيبك ال disassembly بتاع ال binary بتاعك .

---

يعني لو جيت كتبت كذا :

arm-none-eabi-objdump.exe -h app.o \$

ده هيطلعلي ال headers بتاعت ال app.o يعني هيطلعك ال sections سواء text أو data أو bss وكمات هيطلعك سكاشن بتاعت ال Debug طب دي طلعت ليه ؟ عشان احنا كنا حاطين -g فوق وده معناه طلعتي سكاشن من ال binary اللي طالع عشان فيها debug information .

---

ال debug information دي بتساعد ال debugger اللي هو ال gdb لما بييجي ي debug بيعرف ي resolve المكان اللي واقف فيه ك binary عند line كام في ال C ويعرف يجيبك أسامي ال local variables وأسامي ال symbols .

---

ده ال debugger ده حلو يا بشمهندس ! اه حلو في عملية ال debug بس لو انت هتطلع منتج فانت مش بتستخدم ال debug انت بتطلع release يعني بتطلع binary without debug لأنك هتخطه في product انت مش بتعمل debug ليه خلاص .

---

طب ايه السكشن بتاع comment ده ؟ هي السكاشن اللي ملهاش مكان في الحاجات اللي فاتت دي .

---

ال data section زي مانت شايف ال size بتاعه 64 يعني فيه hexa 64 ودول عاملين 100 في ال decimal وطبعاً أنت فهمت دلوقتي ان ال 100 دي بتعبر عن ال 100 بايت بتوع ال [string\_buffer[100 وهي متعرفة على إنها global وما دام هو global variable فهو بيتخزن في ال data memory وال data memory اسمها ال data section في ال sections بتاعت ال binary وال data section كدا فيها 100 بايت وال 100 في ال hex يعني 64 .

---

شوفت بقا دي ميزة لما انت تربط كل حاجة ببعض .. تمام عرفنا ال size ومبسوطين منه .. يعني ايه بقا ال VMA وال LMA ؟

ال VMA ده بيعبر عن حاجة اسمها Virtual memory address of the output section وال LMA بيعبر عن ال Load memory address of the output section .

---

بص عشان تفهم الحنة دي .. ال LMA يعني مكانك فين لما تعمل burn يعني وانت بتعمل burn لل obj بتاعك مثلاً ( لو انت بتهزر ) هو هيتخط عند address كام ؟ هو ده ال LMA .

---

ال VMA هو لما أنا بعد كدا بقا بيقا عندي software ثاني ي copy ال section ده من المكان اللي هو فيه يحطه في مكان ثاني المكان الثاني هو ال Virtual Address بتاعي .. يعني ال Virtual address هو مش ال Physical Address اللي أنا هتخط فيه دلوقتي هو ال Address اللي أنا شايفه في المستقبل ه copy من المكان اللي أنا اتخطيت فيه وأروحله .

---

ومن هنا بقا جات كلمة Relocatable image .. ايه هي بقا ؟ حضرتك خلي بالك ال object اللي طالع ده relocatable image يعني معندهوش Addresses ما يعرفش ال Addresses ما يعرفش البوردة اللي هينزل عليها .. امال مين اللي هيظبطه ال Addresses بتاعته ؟ ال Linker طب وال linker يعرف ال Addresses بتاعته منين ؟ بتاعت ال SPECS اللي هينزل عليها ؟ من ال Linker Script .

---

بيقا ال object اللي طالع معندهوش معلومة هو هيتخط فين ، فهو طالع ب Addresses Virtual أو Addresses كلها وهمية ببسموها Relocatable image لأنها image لسا مش عارفة مكانها ومش عارفة هت locate فين .

---

فهييجي ال Linker وهيكون based على ال Linker Script هيغير ال Addresses دي According لل Linker script اللي انت هتكتبه ، طب وانت هتكتب ال Linker script بناءاً على ايه ؟ .. عيب بقا انت مش تلميذ .. انت استاذ دلوقتي .. هتكتبه based على ال SPECS .

---

فانت في السكاشن لما تلاقهم كلهم ال VMA بتاعهم عند 0000 0000 فده مش معناه إنهم هيعملوا overwrite على بعض .. هما كلهم عند ال 0 لأن كلهم Relocatable image يعني لسا مخدوش ال addresses بتاعتهم .. يعني مفيش حد هي run ال object ده لأن هو لسا ما اتعملوش Linking .

---

## سؤال انترفيو

لوجه سؤال في انترفيو .. ليه ما ينفعش تاخذ ال object وتحطه ؟ هتقوله حضرتك أخده وأحطه ازاي ؟ ما ال Addresses اللي فيه وهمية كلها أصفار لإن ده Relocatable image ده لسا هياخد معلومة ال addresses بتاعته وكل section هيتحط عند address كام من ال Linker وانت عايزني اخده من قبل ما أعدي على ال Linker وأحطه ! طب ما كدا هيطلع حاجات كلها هت overwrite على بعض .

---

طب بشمهندس أنا مش عايز ال debug sections دي أنا عايز السكاشن كإني بطلع product هعمل ازاي بقا ؟ كل اللي هعمله إني هشيل ال -g من ال command بتاع ال compile وهعمل compile تاني .

`arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . uart.c -o uart.o $`

وهكذا من ال app.c

---

وكدا هنبقا شيلنا ال debug information من ال sections فكدنا انت هتخطه على ال product عادي بس مش هتعرف تعمل debug ليه .

---

ففيه ناس بتيجي تقولك بشمهندس ال size كبير قوي وليه عامل كدا ؟ مانت فاتح ال debug حضرتك ومطلع size كبير وزعلان ! افتح وريني السكاشن اللي عندك ، ما فيها كلها سكاشن debug أهو واخده size فانت تبقا انت فاهم فانت اعمل release تتمسح السكاشن دي .

---

## (Executable file sections (.data, .bss and rodata

### سؤال انترفيو

احنا ليه معندناش rodata سكشن في الفايل ده ؟ لإن مفيش حاجة const عندي وده واحد اتسأل فيها في انترفيو قيل كدا .. فيعمل ايه عشان أطلع rodata سكشن ؟ ما تخافش من حاجة افتح الكود وعرف حاجة const دلوقتي ويكون global .. بشمهندس بالبساطة دي ! ايوا ما ال Embedded كدا يا جماعة الدنيا بسيطة أهو .

---

الموضوع بسيط .. ال rodata ده ال section اللي موجود في ال Flash memory ال read only data اللي فيه ال constant variables بتنحط ويتكرت ليها rodata section علطول ، فالموضوع بسيط خالص .

---

فيه واحد قبل كدا قعد 3 ساعات ونص في أسئلة في السيشن بتاعت النهاردة بس .

---

## Global Variables

ال Global variables دي يا جماعة تتحط فين ؟ في ال data section يا بشمهندس .. لأ .. ال global variables بتتحط في ال data section اللي معمول ليها initialization .

لكن ال uninitialized بيتعمل ليها initialize ب 0 وتتحط في ال bss عن طريق ال startup وناس كتير بقا سألت السؤال اللي بيلعبوا فيه على الفهم قوي .

### سؤال انترفيو

قاله يا بشمهندس ال bss موجود فيه ايه ؟ قاله موجود فيه ال global variables ال uninitialized ويحصلها initialization ب 0 وتتحط في ال bss section .

قاله وال bss section موجود فين ؟ قاله موجود في ال Flash memory يا بشمهندس .. غلط مش موجود في ال Flash memory .. ليه ؟ بالمخ كذا ال bss ده هو عبارة عن ايه ؟ نقول مثلاً 100 بايت لو احنا معرفين array of characters ومش عاملين initialize ليها فكلني عندي 100 بايت أصفار .

ليه أحجز مكان في ال Flash memory ل 100 بايت أصفار ؟ طب منا عندي معلومة إن هما 100 طب ما أنا أخلي ال startup at runtime لما يشتغل يروح يحجز مكان في ال RAM ب 100 بايت ويحط فيهم أصفار علطول ويريح دماغه .

ليه أحجز مكان في ال Flash ب 100 بايت أصفار وأعملهم copy ؟ بيقا أنا كذا بهزر .. ده أنا كذا ضايع يعني .. طب مانت في ال startup علطول احجز 100 بايت في ال RAM وخليهم بأصفار .. خلصت .

فخلي بالك من السؤال الرخم ده .. ال bss ده مش بيطلع أصلاً في ال executable اللي طالع .. ليه ؟ لأن هو مش بيطلع في ال Flash أصلاً هو مش موجود في ال Flash .. ده هو انت بمعلومة ال size بتاعته انت بتبقا عندك معلومة bss size بكام وانت كاتب startup code اللي هو ال executable code ده بيروح at runtime يعني لما يشتغل بيروح يحجز مكان في ال RAM ب 100 بايت ويحطهم initialized ب 0 ، بس كذا .

طب يا بشمهندس ليه معملتش كذا في ال data ؟ لا ما أقدرش أعمل كذا في ال data .. ال data لو هي مثلاً 100 بايت هروح أحجز في ال RAM أثناء ال runtime ب 100 بايت وأحط فيهم ايه حضرتك ؟ ما أنا مش عارف ال initialization بتاعهم وده عشان هما already فيهم values فهما ليهم سكشن في ال ROM وأنا بعملهم بعد كذا كوبي للرام .. أما ال bss فهو مفيهوش values فأنا أثناء ال runtime بحجزله مكان وأحط فيه أصفار .

بيقا السؤال ده خلي بالك منه .. ال bss مش موجود في ال ROM .. هو بيتحجز أثناء ال runtime عن طريق ال startup مكان في ال RAM ويتعمل initialize ليه بأصفار .

خلي بالك إن اللي بيتحط في ال data section غير ال global هما ال static .. ال static بيتحط في ال data memory ودي يعني ال data section .

---

## Read-only Data

دي ال const variables وال rodata دي ما بيحصلهاش modification عشان كذا ما بننقلهاش من ال ROM لل RAM .

---

## سؤال انترفيو

ليه ال rodata ما بننقلهاش من ال rom لل ram ؟ زي ال data section ؟ لإن ال data section هي initialized و can be changed at runtime أما ال constant فهو المفروض ما يتغيرش at run time فبسيبها زي ما هي في ال Flash وال Read Only Memory .

---

فال rodata ده constant يعني read-only memory فأنا ليه هنقلها وأحطها في داتا متغيرة في ال RAM ؟ هو أنا بنقل وخلص ؟ بضيع instruction at runtime عشان أنقل وخلص !

---

تمام بيقا احنا نقدر نعرّف concept جديد اسمه ال Load / runtime location .

---

## Load/runtime location

يعني فيه حاجه اسمها load location وفيه حاجه اسمها runtime location .. ال load location يعني انت وانت بتعمل burn حطيت فين ال section بتاعك .. ال runtime location لأ بقا .. دا انت بعد ما عملت ال burn وال startup بتاعك قام وبدأ ال processor يشتغل و ي fetch decode execute فيه .. أثناء ال runtime انت بدأت تنقل حاجات .. الحاجات دي بنسبها ال runtime location .

---

ال runtime locations انت اللي بتكون كاتب ال software يعمل كذا .. يعني انت لو مكتبتش كذا في ال startup مش هيعمل كذا .

ال load location يعني لسا مفيش runtime يعني الباير بتاع البوردة مقفول .

---

## سؤال انترفيو

## Fill it by yourself

الجدول ده بيلعبوا عليه في كل ال MCQ في الشركات ( هتلاقيه موجود في ال slides )

خلي بالك لما بقول **local static** يعني هي موجودة في **scope within** ال **function** بس مش موجودة في ال **stack** امال موجودة فين ؟ في ال **data memory** فينفع كل مرة ألاقي ال **values** بتاعتها ما اتمسحتش .

---

ال **Global initialized** أو ال **Global static initialized** أو ال **Local static initialized** دول هيتحطوا فين ؟ دول هيتحطوا في ال **data** . وال **data** . سكشن ده بيتحط في ال **ROM** أثناء ال **Load time** وبعد كدا عن طريق ال **startup** أثناء ال **runtime** بننقله من ال **ROM** بنحطه في ال **RAM** .

---

بيجي في ال **MCQ** يقولك كدا .. ال **Global static initialized variables are located at runtime location in**

[ RAM - ROM - nothing - all ]

معظم الناس بتختارها **both** لا انت افهم .. ال **global initialized** أو ال **static** موجودين في ال **data** . سكشن وال **data** . سكشن دا بيبقا موجود في ال **binary** النهائي .. لما انت بتعمل **burn** يعني انت بتعمل أثناء ال **load location** يعني بتحطه فين ؟ في ال **ROM** بعد كدا لما ال **software** يشتغل هيبجي ال **startup** ينقل ال **data** . ده من ال **ROM** يحطها في ال **RAM** .  
عشان كدا دي اسمها **runtime location** .

---

يا جماعة افهموا الحتت دي لإنهم بيلعبوا فيها قوي .

---

ال **Global uninitialized** أو ال **Global static uninitialized** أو ال **Local static uninitialized** بيتحطوا فين ؟ دول بيتحطوا في ال **bss** . سكشن عشان هما **uninitialized** طب ال **bss** لما باجي أعمل **burn** بيتحط في ال **load location** فين ؟ يعني وانت قافل ال **runtime** بيتحطوا فين ؟ ده مش موجود يا بشمهندس في ال **load location** أصلاً .. تخيل كدا أنا أحجز في ال **Flash memory** أصفار ! يعني أضبع ال **Flash memory** على أصفار ! بيبقا أنا بهزر .. أنا لما باجي بحجز حاجه بحجز قيم فهي أصلاً أثناء ال **load location** مش موجود ال **bss** .

---

ده ال **startup** أصلاً لما ي **run** يحجز مكان لل **bss** . ويعمله **initialized** ب أصفار بناءً على معلومة ال **size** بتاعته في ال **RAM** أثناء ال **runtime** .. يجيلك بقا في بعض الشركات وخصوصاً فاليو :

(**bss section exists on ROM ? (True/False**

طبعاً **False** ال **bss** مش موجود في ال **ROM** هحجز في ال **ROM** أصفار ازاي بس .

---

ال **local initialized** أو ال **local uninitialized** أو ال **local const** ودي **trick** حلوة قوي جات في شركة محترمة .. ال **local const** دي .

---

هو الحمد لله إن احنا جنبنا سيرة حاجتين احنا فاهمينهم ، ال **local** سواء **initialized** أو **uninitialized** فهو مش موجود في أي سكشن أصلاً يا بشمهندس ، ليه ؟ لأن ال **local** دول أصلاً يا بشمهندس بيتكرتوا **at runtime** يعني انت بتحجز **local** **variable** في ال **stack** صح ؟ اه في ال **stack** .. هل ال **stack** ده موجود في ال **ROM** ؟ بيبقا احنا بنهزر وما اتعلمناش **C**

أصلاً .. ال **stack** أصلاً ما اتركبتش في ال ROM دا بيتمعله creation لما الفانكشن دي بتتندة يعني أثناء ال runtime .. بيقا دول ال **runtime location** بتاعهم هيكون في ال RAM في ال stack .

---

ال **trick** هنا ان ال **local const** بتتعامل معاملة ال **local** العادي .. يعني في الآخر هو في ال runtime location هيكون موجود في ال RAM في ال stack .

---

## سؤال انترفيو

بشمهندس دلوقتي عندك **local variable** موجود في ال **data** . سكشن ولا موجود في ال **bss** . سكشن ومعموله **initialization** ؟ .. طبعاً انت تعرف لما تتسأل سؤال زي ده ان اللي قدامك ده بيحبك حب التنين واللي هو عايز يغلطك بأي شكل .

تقوله يا بشمهندس حضرتك انت بتهزر ؟ ال **local variable** ده بيتخزن في ال stack وال **stack** بيتخزن في ال RAM at runtime ايه اللي دخل ال **bss** ؟ وايه اللي دخل ال **data** . ؟ بس كدا .

---

ال **Global const** ده بقا موجود في ال **rodata** وال **rodata** دي موجودة في ال ROM وهو ده ال **load location** بتاعها .. وبالنسبة لل **runtime location** فهي مش موجودة لأن دي **read-only data** فأنا هحطها ليه في ال رام ؟ أكتب ليه في ال **startup** وأنقلها من ال ROM لل RAM ؟ .. هو مفيش حاجه تمنعني إن أنا أعمل كدا .. بس مفيش حاجه تقولي إنه منطقي إني أعمل كدا فأكيد مش هحطها في ال **runtime location** .. هي موجودة في ال ROM وهستخدمها طول عمري في ال ROM .

---

بيقا الجدول ده لازم تكون فاهمه كويس جداً ، عشان بيلعبوا عليه لعب كثير قوي فتبقا انت مصصح كدا ومركز .

---

## Let us return again to navigate the .obj file

تعالى نجيب ال **disassembly** بقا بتاع الفايل ، هنكتب ال **command** ده :

```
arm-none-eabi-objdump.exe -D app.o $
```

هيطلعك كلام كثير قوي في ال **terminal** ومش هتعرف تقراه ، فاعمل ليه **dump** في فايل تاني :

```
arm-none-eabi-objdump.exe -D app.o >> app.s $
```

---

روح افتح ال **app.s** ده بقا هتلاقي فيه ال **bss** . سكشن موجود عند **address 0** وبعد كدا بقا كل **instruction** واخذ 4 بايت .

هتلاقي فيه **binary** اللي هو ال **op code** وهتلاقي على اليمين ال **instruction** اللي بي فصل ال **binary** ده .

---



ال source code اللي انت بتكتبه بعد ما بتعمله compilation بيطلع طبعاً السكاشن زي ال data. وال text. والباقي ويبطلع حاجه اسمها ال debug information ويبطلع حاجه اسمها ال symbol table .

---

يعني ايه كلمة symbol ؟ بص اسم ال function بيكون symbol واسم ال variable بيكون symbol وال symbol هو عبارة عن address .

---

ال Linker بيمسك ال symbol table بتاع كل object ويشوف مين فيهم ما اتعملوش resolved ويعمله resolved مع الثاني .. يعني لو أنا كان عندي ال symbol اللي هو بتاع اسم الفانكشن ده معمول ليها prototype مثلاً ومش معايا ال definition بتاعها فطلعلي في ال symbol table إن ده unresolved ، يعني ايه ؟ يعني ملهوش مكان يشاور عليه فييجي ال Linker ياخذ من object ثاني من ال symbol table بتاعه إن نفس الفانكشن دي موجودة ومعها ال definition .. فيبدأ ي resolve الاثنين مع بعض .. يعني يشوف ال address بتاع ال definition ده ويحطه هنا .

---

هو احنا أصلاً يا جماعة من ساعة ما بدأنا احنا بنعمل ايه ؟ احنا بنعمل Lab .. احنا المحاضرة بتاعت النهاردة عايزين نعمل Lab1 بس احنا بنتعلم أثناء اللاب واتعلمنا دلوقتي في اللاب إن احنا عملنا حاجتين .. عندنا البوردة وعندنا ال Cross toolchain كتبنا ال C Code files خلاص احنا كدا ميه ميه .. ناقصلنا ايه ؟ ناقصلنا نكتب ال startup وال linker script .

---

ولو عايزين بقا رفاهية والحلاوة كلها نعمل ايه ؟ ن automate الكلام ده كله في حاجه اسمها Makefile وده اللي هنعمله دلوقتي .

---

## Let us write startup code file

أول ما بنفتح الباور لو فيه BootROM هيشغل هيروح يلود حاجه يحطها لك في ال RAM وبعد كدا يروحك عند ال entry point .. ال entry point دي المفروض تحط فيها ال reset section اللي موجودة في ال startup بتاعك عشان بعد كدا تعمل الحاجات وتروح لل main .

---

طب بشمهندس أنا البوردة بتاعتي مفهائش BootROM ، دا انت الدلع كله ، شوف ال processor بتاعك أول ما بيقوم بيروح عند ال address كام وحطلي ال reset section عند ال address ده .

---

بشمهندس ازاي أعمل الكلام ده ؟ عن طريق ال linker .. بشمهندس طب ايه هو ال reset section ؟ هو اللي موجود في ال startup .. طب يعني ايه startup ؟ هو ده اللي هناخد دلوقتي .

---

## C Startup



ال startup هو الكود اللي بيشتغل قبل ال main وال **startup** بي **depend** على ال **target processor** لأن هو بيعمل initialize أصلاً لل target processor .. أهم initialization وأهم حاجة ال stack pointer مثلاً ، ما ال stack pointer هو عبارة عن ايه ؟ هو register inside the processor .

ال startup ممكن بيقا assembly وممكن بيقا C ونيجي هنا بقا ، ايه ده ! بشمهندس ازاي ال startup بال C ؟

## سؤال انترفيو

يا بشمهندس هو ينفع ال startup يتعمل C ؟ ترد تقوله ايه ؟ نيجي نقوله كدا بشمهندس هو ال startup المفروض بي run قبل ال main والمفروض هو من ضمن عملياته إنه يعمل ال stack pointer عشان نعرف نروح لل C صح ؟ اه ، بيقا المفروض يا بشمهندس ده assembly فازاي أنا هخليه يعمل الكلام ده بال C !

الجواب بتاع البشمهندس ده في الشركة هو جواب منطقي وصح ولكن كان فيه معلومة دنيئة جداً في نوع من أنواع ال processors اللي هو ال Cortex-M في ال ARM إن ال stack pointer بتاعه بيتعمل عن طريق ان انت بتخط ال entry point .. بص .

ال processor بتاع ال Cortex-M .. هناخد ال ARM يا جماعه ونفهم الكلام ده .. احنا قولنا أي processor ليه entry point يعني المكان اللي بيقوم عليه في ال Flash memory عشان يشتغل ، كان ال SPECS بتاع ال processor ده بيقوله ال entry point اللي بيقوم عليها ده ال address اللي المفروض ياخذ منه أو يلاقي فيه داتا الداتا دي بتعبّر عن address اللي بياخذها وي set بيه ال SP بتاعه وبعد كدا بيروح لل address اللي بعده .

تاني .. ال Cortex-M Processor أول ما بتفتح ال power بتاعه بيروح لل entry point ولنفترض إنها 0x10000 المفروض انت كنت تحطه ال assembly code في الحنة دي ، لأ ال SPECS بتاع ال processor ده بيقولك نوعه إن لما يروح لل entry point ما تحطش instruction ، امال هحط ايه يا بشمهندس ؟ هيقولك تحطلي global data يعني حطلي value ، ايه ال value دي يا بشمهندس ؟ هتحطلي address .. بيقولك بقا في ال Manual sheet بتاع ال processor ده إن هو بياخذ ال address ده ويحطه لل SP لل Stack Pointer Register بتاع ال Processor هو اللي بيحطه automatic من غير مانت تقوله حطه وبعد كدا بيروح لل Address اللي بعده بقا يبدأ يقرأ assembly ويبدأ يشغله .

بشمهندس يعني أنا مش محتاج أكتب startup يروح ياخذ ال SP ويقول إنه equal ال address الفلاني ؟ لأ ينفع تعمل كدا عادي ، بس هو أصلاً قايلك أنا عندي الميزة دي لأنني أول ما بفتح بروج لل entry point وال entry point بتاعتي لما بلاقي فيها address باخده أحطه أنا automatic في ال SP بتاعي .

طب يا بشمهندس هي دي تفيدني في ايه ؟ تفيدني إن فيه ناس عملت ال startup.c وخلت بداية ال startup اللي هو c. ده محطوط عند ال address اللي بعد ال entry point وحطت في ال entry point ال address بتاع ال stack فايه اللي حصل ؟ ال processor أول ما قام راح لل entry point خد ال address اللي انت حاظه وحطه لل SP بتاعه ، بعد كدا راح لل address اللي بعده لقي أول function في ال C مكتوبة في ال startup بتاعك فشغل ال startup.c .

---

طبعاً أنا لما قعدت أفهم البشمةهندس اللي عمل الانترفيو الكلام ده كله قالي أنا حتى لو اتقبلت في الشركة أنا مش هروح ، وهو اتقبل وراح بعد كذا عادي .

---

الكلام ده كله هناخد في ال ARM لسا خلي بالك ، بس انت لازم ت set الحاجه قبل ما تدخل على ال C ومن ضمن الحاجات اللي لازم تعمل set ليها هو ال stack pointer فمعنى كذا إن ال **startup** لو هو اللي هيعمل ال **set** لل **stack pointer** فهو لازم **assembly** بيقا معظم ال startups بتكون **assembly** لأن هو اللي بي set ال **stack pointer** ويعمل عمليات قبل ما بيخس على ال **main** اللي هو ال C والحل اللي أنا بقوله ده حاجه عشان ال **processor** ده بي support ال **feature** دي .

---

بيقا ال **startup** هو اللي بيقا قبل ال C وال **assembly** بتاع ال C بيكون فيه **stack** فال **startup** لازم يكون مهياً أصلاً ال **stack pointer** بتاع ال **processor** .

---

ال **startup** من ضمن عملياته فيه حاجات كتير قوي .. بس الحاجات المشهورة أنا جمعتها لك ، أول ما بيبدأ بي **disable all interrupts** ، ليه ؟ لأن لو حصل **interrupt** وانت شغال في ال **startup** هيروح لل **handler** ، طب مانت لسا أصلاً بت **initialize** الدنيا فمينفعش تروح ت **handle interrupt** وانت بت **initialize** الدنيا فيقولك بقا في ال **startup** اقلبي ال **interrupt** وأول ما تيجي في آخر ال **startup** تيجي عامل **enable** لل **interrupt** ثاني لأنك خلاص بقا هتكون عملت **initialization** .

---

دلوقتي معظم البوردرات اللي نازلة يا جماعة بيكون **by default** ال **interrupt controller** معمول أصلاً **disabled** ، يعني **by default** وانت شاربي البوردة ال **interrupt** بيبقا مقفول إلا لما انت تيجي تفتحه بايدك فبقا خلاص الناس مش محتاجه تعمل العملية دي ، بس لو انت اتعاملت مع بوردة ال **interrupt** بتاعها **by default** مفتوح لازم تعمل بايدك العملية دي .

---

برضو ال **startup** بيعمل **create** لحاجه اسمها ال **vector table** ، ايه ال **vector table** ده ؟ .. لما ييجي **interrupt** من **module** يروح لل **interrupt controller** يودي له لل **processor** فال **processor** بيسيب كل اللي في ايديه ويروح ل **address** معين في ال **Flash** ، ال **address** ده كان محطوط في ال **SPECs** على حاجه اسمها ال **interrupt vector table** وده **table** بيبقا فيه ال **interrupt** الفلاني رقم 1 مثلاً بيروح ل **address** كام في ال **Flash** .

---

ال **address** اللي في ال **Flash** ده المفروض يلاقي عنده ان انت بتقوله **ISR** بتعمل **branch** لفانكشن **ISR** يعني فساعتها بيسيب اللي في ايديه ويروح لل **ISR** اللي انت معرفها في ال C وينفذ اللي جواها .

---

طب بشمةهندس مين اللي قال إن عند ال **address** ده حطلي إنه هو ي **branch** للفانكشن اللي اسمها **ISR** دي ؟ أنا اللي قولتله كذا .. يعني أنا اللي كتبت **assembly** في الحتة دي ، طب الحتة دي اسمها ايه في ال **embedded** ؟ اسمها **vector** **section** .

---

طب ال vector section ده اللي جاي من ال vector table اللي أنا بقول عند address فلان روح jump للحنة الفلانية في ال C ده مين اللي بيكتبه ؟ أنا اللي بكتبه .. بكتبه فين ؟ بكتبه في ال startup .. بس هو دا اللي أنا عايزك تعرفه في الحنة دي .

---

ال startup برضو بيعمل copy لل initialized data اللي هو ال data . من ال ROM لل RAM .

---

ال startup برضو بيحجز مكان لل bss . في ال RAM وبيحط فيها أصفار .

---

ال startup برضو بيحجز مكان لل stack وبعد كذا يخلي ال stack pointer بتاع ال processor يشاور عليه .

---

ولو أنا عندي dynamic allocation بحجز مكان لل heap .

---

وفي الآخر خالص بيعمل enable لل interrupt بقا عادي ثم آخر حاجه بيروح ينده على ال main .. دي الحاجات ال famous أو ال most-used في ال startup .

---

## Stack

هو بيعمل ايه ؟ هو بيخلي ال processor ال stack pointer بتاعه يشاور على أوله عشان يعمل push و pop بعد كذا كل ما بينادي على C Function .

---

disable interrupts  
initialize stack pointer  
transfer data section from rom to ram  
reserve location for bss section in ram  
create vector section for vector table  
initialize head for dynamic memory allocation  
enable interrupts

## Tasks of startup code

- 1 Disable all interrupts
- 2 Define interrupt vectors section
- 3 Initialize memory and hardware
- 4 Copy data from ROM to RAM
- 5 Initialize data area
- 6 Initialize stack
- 7 Enable interrupts

-8 ( ) Create a reset section and call main

---

## Writing a startup.s file

ال startup ده المفروض هتباصيه لمين يا بشمهندس ؟ هتباصيه لل assembler .. ليه يا بشمهندس ؟ ما هو ال startup ده s. يعني ده assembly .

---

احنا في Lab1 يا شباب هنعمل startup بسيط خالص لإن احنا لسا بنتعلم ، بس Lab2 أوعدكم هنعمل startup تطلع من السيشن دماغك عالية قوي .

---

ايه ال simple startup اللي هنعمله في Lab1 :

هنعمل reset section ينده على ال main بس قبل ما نعمل الكلام ده ي initialize ال stack .

---

ايه اللي هنعمله في Lab2 ؟ هنعمل كل حاجة هن define ال interrupt vector وهننقل الداتا من ال ROM لل RAM وهنجز مكان لل bss وهنروح نعرف مكان لل stack ونشاور عليه وبعد كذا انده ال main ، الكلام ده هنعمله المرة الجاية ان شاء الله .

---

بس النهاردة هنعمل حاجتين سهلين تبع Lab1 عشان نمشي مع بعض واحدة واحدة كذا ، المنظر ده بتاع ال startup طب ايه دا ؟ انساه ، بس انت هتكتب أعقد منه المرة الجاية ان شاء الله ، أنا بس عشان مش عايز أتقل عليك بس لو انت بصيت كذا ، دا ال vector section ، وانت في السي كود لما تيجي تعرف function بتكتب اسم ال function وتفتح brackets ، لكن في ال assembly لو انت عايز تعمل حبة assembly code يبقوا تبع label معين بنسميه section ، بنعرف Label ونحط تحتيه ال assembly بتاعه .

---

فمثلاً ده ال vector section ال vector section ده فيه لما يحصل interrupt يروح ل address معين وساعتها يلاقي branch ل handler وال handler ده C code .

---

ايه ال globl. دي ؟ دي بنستخدمها في ال assembly لما نحب نعرف symbol ، يعني نخلي section معين يتشاف من ال C code أو من اي فايل ثاني برا .

---

ده ال reset section عمل ايه ؟ .. لود instruction ، لود ال address ده في ال stack pointer وبعد كذا بيروح ي branch على سكشن اسمه init\_sect ودي لما راحها طلعت في الاخر عبارة عن ايه ؟ دي عبارة عن function في ال سي كود وفيه بقا بدأ ينقل الداتا من ال ROM لل RAM وبدأ يحجز مكان لل bss ويخليه أصفار ، انسى الكلام ده دلوقتي المرة الجاية هنفهمه ان شاء الله ، وال reset بعد ما خلص ال init\_sect ده راح عمل branch لل main .

---

كل الكلام ده بصينا عليه دلوقتي عادي بس لسا هنتعمق فيه المرة الجاية ان شاء الله ، فسبيك منه دلوقتي .

---

طب احنا بقا عايزين المنظر بتاع النهاردة ، بلا نروح للمكان بتاع فولدر Lab1 ونعمل فايل :

touch startup.s \$

---

هنروح نفتح startup.s ونعمل section اسمه reset ، جوا ال section ده هنكتب ldr وده assembly code في ال arm عشان يلود حاجه يحطها في register ، فأنا عايز أعرف ال address ال stack pointer يعني الود value في ال sp .

---

بص بقا ، من ال memory عندي ال entry point عند 0x10000 ، أنا أكيد هقعد في الحنة دي أحط فيها سكاشن كتير بص عمر السكاشن ما هتوصل لآخر ال memory ، أنا عايز أجي مثلاً عند 0x00011000 دا بيقا ال top بتاع ال stack بتاعي .

---

طب يا بشمهندس انت صح ان انت تعملها عشوائي كذا ؟ .. لأ طبعاً ، إن أنا أعملها عشوائي كذا غلط ، بس احنا لسا بنتعلم ، وأنا بتعلم حاجي في حنة معينة في stage هقولك مش هينفع بقا نمشيها عشوائي ، تعالى بقا ناخذها من ال linker بحسابات ال linker ، فاصبروا معايها واحدة واحدة كذا .

---

فهتبقا كتبنا دول :

: reset

ldr sp, = 0x00011000

ده معناه حطلي ال value دي اللي هي constant فحطيت جنبها = ، هحطها فين بقا ؟ في ال sp

---

بشمهندس أنا ما أعرفش assembly ، ما ده assembly ARM .. مفيش مشكلة ، أنا اللي عايزك تتعلمه ان انت تبقا professional embedded لما تشوف startup ما تخافش ع الاقل تبقا فاهمه ، تعرف تجيب خلاصته الزتونة بتاعته ، مش لازم تكتب ، لإن انت أكيد مش هتتعلم assembly بتاع power pc و ARM و SH4 .. مفيش حد بيتعلم كل ال architectures بس ع الأقل تبقا فاهم ايه اللي بيحصل .

---

بعد كذا bl يعني branch label يعني روح اعمل برانش لل label ده ، ايه هو بقا ؟ ال main فهتبقا كذا :

bl main

---

طب افرض بعد كدا رجع من ال main ، يرجع ايده فاضية ؟ يعني افرض أنا روجت ال main خلصت ، مكننتش عامل while(1) في ال main هعمل ايه ؟ هعمل label ثاني اسمه stop ونحطه في كود ال startup تحت ال reset علطول بس مش جواها ، جوا ال stop ده هقوله :

: stop

b stop

---

يعني ايه ؟ يعني انت رجعت من ال main ، انت مكننتش عامل while(1) في ال main عشان كدا بيقولك في ال embedded حط while(1) جوا ال main ، ليه ؟ عشان مترجعش ، لأن انت لو رجعت في ال reset section هتدخل في اللي بعده اللي هو ال stop وأول ما تدخل فيها هيقعد يعمل برانش على ال stop يعني كأنك بتعمل infinite loop والدنيا هتقف .

---

بشمهندس ناقص ايه هنا ؟ أنا ال reset section ده عايزه بيقا هو ال entry point بتاعتي فأنا عايزه ينفع يتشاف من ال linker script ، فخلي بالك ال reset section ده هيبقا symbol ، مش ال startup.s ده يا جماعه هيخش على ال assembler يطلع object file ؟ واحنا قولنا ان كل object file ليه symbol table .

---

ال symbol table يعني فيه أسامي ال symbols عند addresses فأنا عايز أقول إن ال symbol ده global عشان ينفع يتشاف من ال linker script عشان هستخدمه في حاجه ، فلو انت عايز تعمل ال symbol ده global هتقوله:

global reset.

---

أنا كدا عملت ايه ؟ أنا كدا قولت ال reset section ده خليهولي global عشان يتشاف من الفايلات اللي برا ، بيقا متشاف في ال symbol table يعني .

---

كدا احنا كتبنا الكلام ده في ال startup.s لحد دلوقتي :

global reset.

: reset

ldr sp, =0x00011000

bl main

stop : b stop

---

السؤال اللي ببيجي في الانترفيوهات ، ينفع إن أنا ال application بتاعي ميقاش فيه main ؟ اه ينفع اهو لو غيرنا كلمة main اللي بيعمل branch ليها جوا ال reset دي هيبقا ال application بتاعي بيشتغل عادي باسم الحاجه الجديدة ومن غير main .

---

خلاص انت بقا فاهم دلوقتي ، ليه احنا كنا بنقعد نقول ال main ؟ لإنك انت كنت already مش بتكتب ال startup فأني startup في الدنيا كان بينده على symbol ال main ، بس كدا .

---

الموضوع بسيط ما تخافش ، أنا عايزك توصل لمرحلة كدا تلعب بال embedded ، بس خرينا ماشيين مع ال main عشان احنا شغالين على convention name ما ينفعش نيجي في شركة ونسميها باسم ثاني .

---

أنا كدا خلصت كتابة ال startup تعالى بقا ن compile ، طبعاً عشان ن compile هنروح نستخدم ال assembler وهنقله كدا :

```
arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o $
```

---

كدا احنا عملنا ايه ؟ احنا طلعلنا ال startup.o ، تعالى بقا نتفرج عليه :

```
arm-none-eabi-objdump.exe -h startup.o $
```

---

هيطلع ليك السكاشن وال size بتاعها ، هتلاقي ال .data أصفار لإن مفيش داتا انت مش معرف داتا ، وبرضو ال .bss. أصفار لإنك مش معرف bss ، اللي موجود بس هو ال text ، بس الدنيا بسيطة اهي .

---

ألف مبروووك ، انت عندك دلوقتي cross toolchain ، عندك C Code file ، عندك startup.s ، ناقصلك ايه يا بشمهندس ونبقا خلصنا النهاردة ؟ ناقصلك ال linker script .

---

## Let us write linker script

ال linker script هو مجموعة objects مع بعض يعملها linking together عشان يطلع في الاخر ال output section ، يعني هو خد مثلاً ال Data 3 اللي عندي وطلع Data section جديد ، خد ال 3 bss اللي عندي وطلع bss section جديد ، ونفس الكلام مع ال text .

---

## سؤال انترفيو

بشمهندس هو أنا ينفع أخذ ال data. الأولاني مع ال data. الثاني مع ال data. الثالث وأطلع section جديد اسمه Keroles ؟  
اه ينفع مفيش مشكلة .

ايه ده يعني ينفع سكاشن اخدها مع بعض وأطلعها باسم section ثاني ؟ .. اه ، انت في ال linker script اللي هو Id. ده تقدر  
من خلاله تاخد اي حاجة مع اي حاجة وتطلعها باسم اي حاجة ، ولكن مش مستحب إلا لو فيه حاجة في دماغي .

هي ال objects دي يا جماعه اسمها ايه ؟ relocatable image يعني images مبتقاش عارفة ال location بتاعها فين  
في ال physical فلما خدتها مع بعض ، أنا بحطهم عند addresses هي دي اللي هتتحت على ال physical ، يعني ال  
binary اللي طالع في الآخر هيقا عنده ال LMA وده ال Address اللي هيتحت فيه في ال physical .

لكن ال VMA كان هو ال address ال virtual يعني بعد ما بخلص ال physical يعني أثناء ال runtime حاولت **أنقله**  
وأحطه في مكان ثاني ، المكان الثاني ده هو بالنسبالي ال virtual address .

فكريين اللي احنا طلعهنا ، احنا طلعا حاجة اسمها VMA و LMA .. ال LMA هي ال load time address يعني ال  
binary ده هيتحت فين في ال flash لما ال processor يكون نايم في ال physical address .

أما ال VMA ، أثناء ال runtime ال software بتاعي لما بييجي ينقل حاجة ويحطها في مكان ، المكان ده فين ال VMA بتاعه  
؟ يعني هيتحت فين ك virtual ، هو ليه virtual يعني لسا ما اتحطش ، أثناء ال runtime هيتحت .

بشمهندس أنا بدأت أتغبط وبدأت كذا أخاف ، لا اصبر كذا وهنظبط الدنيا .

**يبقا ده كذا معناه ان ال linker script ده هو based على ال target microcontroller ، طبعاً هو based عليه ، مش هو**  
بيقولي الحاجات بتتحت فين ؟ .. بيقا لازم يكون عارف ال addresses في ال microcontroller عاملة ازاي ، مش محتاجة  
فكاكة .

ال linker script بياخد option اسمه -T

بشمهندس ، ال linker script ده عقدتنا ، بنخاف منه ، هو ايه ال linker script ؟ .. طب ايه رأيك بقا ان انت اللي هتكتب ال  
linker script ، طب ازاي يا بشمهندس ؟ .. عشان تكتب linker script لازم تعرف ان ال linker script ده ليه  
commands .



---

ال commands دي زي ال ENTRY, MEMORY, SECTIONS, Location Counter, Symbols, ALIGN, KEEP, INPUT, OUTPUT ، النهاردة هتأخذ حبة حاجات والمرة الجاية هنكمل ان شاء الله

---

أول حاجة انت بتكرت فايل بتسميه اي اسم .ld ، ال text اللي انت بتكتبه في الفايل ده بيبكون based على commands ، ال commands دي بيبكون ليها syntax .

---

## ENTRY Command

هروح في الفولدر بتاع Lab1 أعمل فايل أسميه linker\_script.ld ، أول حاجة .. بشمهندس ايه هو ال entry ؟ .. ال entry كالاتي ، ده ال syntax بتاعه :

(ENTRY (symbol

بتيجي في أول الفايل تكتب ENTRY وتباصيله اسم ال symbol .

---

يعني انت بتوصف الفايل بتاعك ده ال executable اللي طالع بال debug information إن فين المكان بتاع ال entry ، يعني اسمه ايه ال symbol اللي بيدأ بيه ؟ .. بشمهندس مش انت كنت معرف في ال startup فيه reset section وال reset section ده is a symbol يعني أنا عايز إن ال processor ده بيدأ فيه أول ما يقوم ، بيقا المفروض أعمل ايه ؟

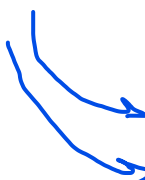
---

هروح أقول لل linker script :

(ENTRY (reset

يعني ده كذا ظبطلي إن ال processor لما يبجي يقوم ، ولنفترض إن البوردة اللي احنا شغالين عليها ال entry point بتاعها كان عند 0x10000 ده كذا معناه ان ال processor أول ما يقوم هيقا ال reset ده عند 0x1000 ؟ .. هقولك طب أنا هسألك بزمته كذا ، هو أنا لو قولت (ENTRY (reset انت شايف ال linker script عرف معلومة ال 10000 دي ؟ .. مفيش حاجة تقوله على ال 10000 دي خالص .

---



امال ايه بقا حته ال ENTRY دي ؟ .. ال ENTRY دي عشان تعرف ال debugger وهو بي debug ان انت بدابتك المفروض هي ال reset ، ولكن مش هي دي المسؤولة لما تبجي تعمل burn تحرق ال application ده يحط ال reset عند ال address ده ، امال ايه اللي بيحط ال reset عند ال address ده اللي هو 0x10000 ؟ .. ده حاجة في ال linker script اسمها ال locator لسا هنخش فيها ال command اللي جاي .

---

أما ال ENTRY ده هو معلومة بس عشان ال executable file بتاعك ده لما تبجي في ال binary utility تقرا المعلومات بتاعته ، بيقا ال ENTRY keyword is used for defining the entry point لل application و this information will be in the final elf file ، يعني ال information دي موجودة في ال header ولكن دي لا تؤثر على

ال section ده لما بييجي يتحرق في ال memory يتحط عند 0x1000 ، يعني مفيش حاجة بتقول إن ده يتحط عند 0x1000 ملهناش اي علاقة يعني .

ال reset section ده ال most used في عالم ال embedded إن ده اللي نبدأ عنده ، ال debugger بيستخدم المعلومة دي عشان يعرف فين بدايته ، يعني المفروض ي debug أول break point يحطها فين .

ال ENTRY دي not mandatory ، يعني أنا لو معملتش لل ENTRY .. linker script عند ال reset مش هيحصل حاجة ، ايه ده يا بشمهندس يعني ينفع ؟ .. ايوا يا جماعه انت كبر مخك ، هو ايه اللي يفرق مع ال processor ، ال processor يفرق معاه إنه لما يقوم ويروح ل address 0x10000 في ال Flash memory يهيمه معلومة ENTRY ؟ .. لا مش مهم .

هو نفسه يلاقي فعلاً ال assembly الأولاني موجود عند 0x1000 اللي هو ldr sp اللي هو بداية ال reset ، طب دي تتحط ازاي ؟ .. دي تتحط في ال command اللي جاي ، إن احنا نقول لل linker script وانت بتطلع ال binary النهائي حطلي ال ldr ده عند address 0x10000 فلما بييجي ال burner يحرقه يحط ال instruction ده عند 0x10000 .

أما ال ENTRY command ده بس just for information للدلع يعني ، لل debugger والحاجات دي ، فعشان كذا هيا not mandatory ، بس احنا خيلنا professional .

فهاجي عند ال linker\_script.ld وأجي في أوله عشان دي معلومة بيستخدمها بس ال debugger وأي حد بعد كذا بي hack ال binary بتاعي يعرف يتفرج على المعلومات اللي أنا عايزها بس فهقوله :

(ENTRY (reset

## MEMORY Command

ده بقا بيستخدم يا جماعه في ال linker script عشان أوصف فيه أي memory عندي ، يعني ال memory command ده بوصف فيه ال memories اللي موجودة في ال microcontroller أو ال SOC ، ايه ال syntax بتاعه ؟

MEMORY

}

name (attr) : ORIGIN = origin, LENGTH = length

name : o = origin , l = length

...

{

ال name ده الاسم بتاع ال memory ممكن أسميه أي اسم أنا عايزه ، بس الأفضل يكون على أساس ال SPECS ، يعني انت بتوصف أسامي لل memory accoring to SPECS .

---

وفيه attributes ممكن كمان تعرّف attribute ، ال attributes بتعرّفها يا read-only يا read-write أو فيه كذا حاجة كمان ممكن تعرفها بيهم عادي .

---

بعد كذا بتقول كلمة origin ، يعني ال memory دي بتبدأ من عند فين وال size بتاعها كام ، وساعات ممكن تكتب o بدل ORIGIN و ا بدل LENGTH .

---

طب بشمهندس ازاي ؟ اديني مثال ، في بعض ال microcontrollers عندك ROM وال ROM دي مقسمها لحايتين جزء عايز تخليه لل vector جزء لل ROM وجزء لل RAM وجزء لل SRAM ، فساعتها تعمل ايه ؟ هتوصف ال memory بتاعك في ال startup تقول كالاتي :

MEMORY

}

vect : o = 0 , l = 1k

rom : o = 0x400, l = 127k

ram : o = 0x400000, l = 128k

sram : o = 0xfffff000, l = 4k

{

---

في حالتنا احنا بقا في ال Lab1 هنشتغل على memory واحدة بس ، في ال Lab2 بقا هنشتغل ان شاء الله على اتنين أو ثلاثة memory هيبقا الدنيا اصعب من كذا ، ال linker بتاع ال Lab الثاني محترم فخليني كذا ابدأ معاكم واحدة واحدة .

---

يلا نكتب ال linker script بتاعنا :

(ENTRY (reset

MEMORY

}

Mem (rwx ) : ORIGIN = 0x00000000 , LENGTH = 64M

{

ال rwx دي attributer معناه read write execute يعني أنا عايز أعمل الثلاثة دول في الميموري دي .

---

## SECTIONS Command

انت هتاخذ ال section من ال objects مع بعض وتطلع انت سكاشن بأسامي ثانية ، مش فاهم حاجه يا بشمهندس ، تعالى كدا بص معايا :

SECTIONS

}

: Keroles.

}

(text.)\*

{

أنا كدا عملت section جديد هيطلع معانا في ال output اسمه Keroles ، طب ال syntax بتاعه ايه ؟ .. بتكتب اسم ال section ال output اللي انت عايزه يطلع وتحط قبله نقطة وبعده نقطتين فوق بعض ، وتيجي جواه تكتب أسامي ال sections اللي انت عايزها تطلع .

---

يعني مثلاً تيجي تكتب جواه \*.text ، يعني ايه كدا يا بشمهندس ؟ يعني أنا بقوله خدلي ال \*.text من كل ال inputs ، انت عندك object 3 داخلين ال linker عندك ال \*.text , application.o, uart.o .

---

ال \* دي wild card بتستخدم معناها جمعلي كل ال text. مع بعض وطلعلي output اسمه Keroles. واللي هيكون جواه كل ال text. اللي موجودة دي ، ينفع ؟ اه ينفع مفيش مشكلة .. بس أنا هعمل كدا ليه ! ماهو أنا برضو الناس لما تيجي تقرا ورايا هيقولوا Keroles. ده ايه ! فياريت نعمل حاجه تدل على convention name معين ، بس مفيش حاجه تعترض إني أعمل كدا يعني .

---

يعني لو أنا جيت دلوقتي وقولت :

```
arm-none-eabi.ld.exe -T linker_script.ld app.o uart.o startup.o -o learn-in-depth.elf $
```

---

انتوا طبعاً مش هتكتبوا linker في الشركات ، هو بيكون جاي جاهز .

---

لو جينا بعدها وقولنا :

```
arm-none-eabi-objdump.exe -h learn-in-depth.elf $
```

دي عملناها عشان نشوف السكاشن ، هي هتطلعنا وهيطلع لنا كيرلس معاهم ك output section وال text. مش هيطلع معنا ، طب ليه ؟ لإن Keroles ك output section بقا فيه كل ال text. بتاعت كل ال objects اللي داخلين .

---

احنا دلوقتي عملنا ايه ؟ .. عرّفنا فيه Command في ال linker script اسمه SECTIONS بتكتب فيه كل ال output sections اللي عايزها تطلع وكل output section جواه ايه من ال inputs ، فأنا كإني بقوله ايه ؟ جمعلي كل ال text. سكشن اللي داخلين inputs في كل ال objects ال 3 اللي هما ال app, uart , startup وحطهملي في output section اسمه Keroles.

---

فلما عملت كدا وعملت ال linker script وروحت شوفت ال header اللي طلع في ال executable النهائي ملقّتش ال text. طلع ، ليه ؟ لإن كل ال text. بقا في Keroles.

---

طبعاً حاجه زي كدا يا جماعه احنا نستخدم convention name بتاعت ال embedded ما ينفّش أقول Keroles والكلام ده .

---

أنا بقا براحتي بقا ، بصوا بقا المزاج ، أنا عايز أعمل section اسمه startup ، ليه اسمه startup ؟ عشان عايز أقول ان ال section ده أحطه عند ال address بتاع ال 0x10000 ، مش احنا كل ده عشان نحط ال reset section اللي جوا ال startup عند ال 0x10000 ؟ فأنا خليت ال reset section الموجودة في ال startup تطلع في output section اسمه startup.

---

بس أنا مش عايز ال `startup` ده بيبقا فيه كل ال `text` ، أنا عايز بيبقا فيه ال `text` بتاع ال `startup.o` ، فالكود جوا ال `linker` script هيبقا عامل كدا :

## SECTIONS

```
}
```

```
: startup.
```

```
}
```

```
(startup.o)(text
```

```
{
```

```
: text.
```

```
}
```

```
(text.)*
```

```
{
```

```
: data.
```

```
}
```

```
(data.)*
```

```
{
```

```
: bss.
```

```
}
```

```
(bss.)*
```

```
{
```

```
{
```

---

طبعاً ال `text` بتاع ال `startup.o` اللي هو مين ؟ ال `reset section` ، لإن أنا مكتبتش في ال `startup` لحد دلوقتي غير ال `reset section` فالسطين اللي جواه هما دول ال `text` اللي هيطلعوا ، فأنا كدا في الكود اللي فوق ده قولتله ال `startup.o` خدلي ال `text` اللي فيها طلعهولي في `output section` اسمه `startup`.

---

بعد كدا بقوله طلعهولي `text` سكشن تاني ، جمعلي فيه جميع ال `text` بتاعت ال `software` الباقية بقوله فيها جمعلي كل ال `text` بتاعت كل السكاشن التانية .

---

بعد كذا بقوله عرفلي في ال output section ال data. سكشن وحطلي فيها أي حاجة data.

---

كدا فاضل ال bss. فهقوله جمعلي ال bss. في كل السكاشن في كل ال software في كل ال objects اللي طالعين وحطهالي في ال bss.

---

تعالى نشوف بقا الكلام ده بعد ما كتبناه ايه اللي حصل ، هقوله :

```
arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o startup.o -o learn-in-depth.elf $
```

وبعدها :

```
arm-none-eabi-objdump.exe -h learn-in-depth.elf $
```

---

هتلاقي طلع معاك ال startup. وال text. وال data. ، ومطلعش ال bss. لأن أنا معنديش حاجة uninitialized ، طلعي كمان ال rodata. وال comment. ودول مضايقتي .

---

ال comment. ده يعني أي حاجة موجودة في ال toolchain ملهاش section في أي حاجة من الأساسيين فحطهم في ال comment. ، احنا عابزين نخفيهم بقا ، تعالي نحط ال rodata. مع ال text. يعني بس احنا بنتعلم دلوقتي ، فكأننا عابزين نحط الاثنين دول مع بعض في سكشن واحد ، هو ينفع ؟ اه ينفع ايه اللي يمنع .

---

## SECTIONS

}

: startup.

}

(startup.o)(text

{

: text.

}

(text) \*(.rodata.)\*

{

```

: data.
}

(data.)*
{

: bss.
}

(bss) *(COMMON.)*
{
{

```

---

فكدا ال `.rodata` هنتحط مع ال `.text` في ال `output section` اللي اسمه `.text` ، وخطينا برضو ال `comment` في ال `.bss` ، يا بشمهندس أصلاً مفيش `bss` طلع معنا ! هو مفيش `bss` طلع فتعالى نحط معاه ال `comment` . وخلص وكإن ده تبع ال `bss` . مع إنه مش تبعه بس احنا لو خطينا اي حاجه اسمها `COMMON` ودي معناها اي حاجه ملهاش `section`

---

لو جينا نشغل تاني بقا هنلاقي ال `.rodata` اختفى يا جماعه .

---

طبعاً احنا كدا بناخد ال `input sections` بنعملهم `merge` ونحطهم في `output section` اسمه `.text` . وهكذا بقا .

---

## Location Counter

فيه بقا حاجه مهمة جداً في ال `Linker` اسمها ال `location counter` ودي خطيرة خطيرة ، يعني ايه ال `location counter` ؟ .. ده يعني انت لو عندك `memory` وعمايز تقول مثلاً بشمهندس ال `reset section` حطهالي عند `address 0x10000` .

---

فيقولك فيه علامة ال `dot` في ال `linker script` ، دي لما تقوله `0x10000 =` . كإنك كدا عندك حاجه واقفة عند `0x10000` ، بعد كدا تحط ال `section` بتاعك ، وبعد ما يحط ال `section` اوتوماتيك بييجي بعديها يتحركها ، يعني حاجه عمالة تشاور معاك وتحسب ال `size` اوتوماتيك في ال `linker script` .

---

بشمهندس أنا مش فاهم حاجه ، قبل ما نخش في ال `location counter` لازم نفهم ال `command` ده الأول عشان نعرف نفهم الباقي .

---



## <(lma) AT≥(vma)

فاكرين يا جماعه ان انا قولتلکم کل object طالع منه اتنين address ال vma وال lma ، ال lma ده يعني لما احرق ال software ال section ده يتحط عند address كام في ال flash memory .

امال ال vma ؟ دي معلومة لل address يس لما باجي بحرقه ال Address ده ما بيتحطش ، امال المعلومة دي ايه ؟ عشان at runtime أنا عايز انقله واحطه في مكان تاني هستخدم ال vma .

يعني ال vma مش physical address ، لا ده ال Address اللي نفسي أوصله لما أشغل الكود ، وهو بيستخدم أكثر ك virtual يعني ده اللي بحلم إن أنا أوصله .

عشان تفهم الجزء ده ، تعالى نبص على المثال ده ، خلي بالك المثال ده من Lab2 فمتقلقش يعني من المنظر ، أنا دلوقتي عايز أعمل المنظر ده :

MEMORY

}

RAM (rwx) : ORIGIN = 0x00000000 , LENGTH = 64M

ROM (rx) : ORIGIN = 0x00000000 , LENGTH = 64M

{

SECTIONS

}

: startup.

}

(startup.o(.text

VMA @> LMA <{

: text.

}

(text) \*(.rodata.)\*

{

: data.

```

}
(data.)*
{
: bss.
}
(bss) *(COMMON.)*
{
{

```

---

أنا اهو عندي RAM وعندي ROM ، باختصار كدا ، ال text. بيبقا موجود فين يا جماعه لما نيجي نعمل burn ؟ بيبقا موجود في ال ROM ، فيقولك اعمل الحركة دي :

```

: startup.
}

```

```

(startup.o(.text

```

```

VMA @> LMA <{

```

الحركة دي معناها السكشن ده موجود عند address كام ك VMA أو عايزه بيقا عند address كان أثناء ال runtime ووأنا بعمل burn هيكون عند address كام اللي هو ال LMA .

---

طبعا ال text. سكشن ما بيتحطش في ال virtual عند حته ، يعني هو أثناء ال runtime أنا مش بعمله copy ، ده هو موجود في ال ROM ولما الكود يشتغل أثناء ال runtime هيفضل موجود في ال ROM ، يعني هو ال LMA في ال ROM وأثناء ال runtime هيفضل برضو في ال ROM بيقا ساعتها هقوله كدا :

```

: startup.
}

```

```

(startup.o(.text

```

```

ROM @> ROM <{

```

---

فده معناه إن هو أثناء ال runtime هيبقا في ال ROM وأثناء ال physical time أثناء ال burning هو في ال ROM فيقولك ما دام الاتنين هيبقوا ROM بيقا لو سمحت لو سمحت هتعملها كدا :

```

: startup.

```

}

(startup.o(.text

ROM <{

---

ال text. برضو نفس الكلام ، ال LMA وال VMA هيبقوا في ال ROM فهتبقا كدا :

text.

}

(text) \*(.rodata)\*

ROM <{

---

ال data. هتتخط فين في ال VMA وال LMA ؟ ، أثناء ال burning ال data. موجود في ال ROM فهتتخط في ال ROM ، ال VMA لما الكود بتاعي ك startup يشتغل مش هو هي copy ال data section دي يحطها من ال ROM لل RAM ؟ .. بيقا ال VMA هو ال address اللي نفسي يوصله فيه أثناء ال runtime فهو ال address بتاع ال RAM ، فهتتعمل بالشكل ده :

data.

}

(data.)\*

RAM @> ROM <{

---

ال bss. تتخط فين ؟ ال VMA بتاعها في ال RAM وبرضو ال LMA بتاعها في ال RAM ، يعني هي أنا نفسي تتخط في ال RAM أثناء ال runtime وأنا أصلاً نفسي من دلوقتي تبقا في ال RAM فروحت حطيتها في ال RAM كدا :

bss.

}

(bss.)\*

SRAM <{

---

طب بشمهندس أنا مش فاهمها ، أنا مش عارف أفهمها ومخي قفل ، ما تقلقش هناخدوها المرة الجاية ان شاء الله .

---

أنا دلوقتى بتعامل كإني معنديش غير memory واحدة اللي أنا مسميها Mem ف ما دام أنا مسميها Men فهبيقا بالنسبالي كلهم في ال Mem .

## SECTIONS

```
}  
  
: startup.  
  
}  
  
(startup.o(.text  
  
Mem <{  
  
: text.  
  
}  
  
(text) *(.rodata.)*  
  
Mem <{  
  
: data.  
  
}  
  
(data.)*  
  
Mem <{  
  
: bss.  
  
}  
  
(bss) *(COMMON.)*  
  
Mem <{  
  
{
```

---

بيقا أنا كدا خليتهم ال VMA وال LMA في ال Mem لأن أنا معنديش أي حاجه ثاني ، طب بشمهندس ايه بقا حتة ال location counter اللي انت قولتلي أجلها ؟

---

## Location Counter

ال dot دي بقا أهم حاجة في ال linker scripter ، هي اللي بتظبطلك ال linker scripter ، ازاي بقا يا بشمهندس ؟ بص معايا .. هو انت دلوقتي لما نكتب الكود بتاعنا ، انت بترسم على الورق ال layout بتاعك ، يعني انت عايز تحط ال startup وال startup ده موجود ال reset section جواه عايز تحطه عند 0x10000 ، بعد كذا عايز تحط أي text. بعديه بعد كذا data. بعد كذا .bss. ، بعد كذا عايزين نسيب 1000 بايت ونقول بعدها ان ده ال stack top عشان بيقا ال stack بتاعنا 1000 بايت .

---

بشمهندس أرسم ال layout ده ازاي ؟ يقولك علطول تجري على ال dot وده اسمه location counter ، ليه location counter ؟ لأن فيه option حلو أوي ، ان انت لما تيجي تكتب ال syntax بتاعه بتقوله :

el mkan elli ana wa2ef fih = .

---

فلما أقوله :

; 0x10000 = .

فأنا كذا كإني بقوله ال dot بتاعي واقف عند النقطة دي 0x10000

---

طب انت من غيره كنت واقف فين يا بشمهندس ؟ يقولك انت كنت من غيره واقف عند 0x0 لأنك انت بتقول ان SECTIONS ده كله موجود في Mem وMem ال ORIGIN بتاعها 0x0 ، فلما انت عملت الحركة دي :

;0x10000 = .

فانت كذا بتقوله انا جوا ال Mem دي واقف عند ال 0x10000

---

بعد كذا بقا بتقوله ال output section اللي اسمه startup ده حطهولي عند ال dot ، بالطريقة دي :

: startup.

}

(startup.o(.text

Mem <{

طب هي ال dot بكام ؟ ب 0x10000 ، بيقا كذا ال reset section اتحط عند ال entry point يا معلم ، ما هي مفهائش اهي إلا ال text. بتاع ال startup ال input section اللي جاي من ال startup.o .

---

بعد كذا هعمل ايه ؟ بعد كذا هو automatic بقا هيقعد ي increment ، يعني هيحط ال startup أو ال reset عند ال 0x10000 وبحسب automatic ال size بتاع ال text اللي جواه ، ويجمع على نفسه ويحط ال text. وبعد كذا يجمع على نفسه ال size بتاع ال data. ويحط ال data. وبعد كذا يجمع على نفسه ال size بتاع ال data. ويوصل لل .bss.

---

عايز بعد ما يوصل لل bss. أعمل ايه ؟ أقوله :

. = . + 0x1000 ;

جمعنا عليه في الآخر ال size بتاع ال stack بتاعي اللي هو 0x1000 ، دي يعني 4 كيلو بايت .

---

وبعد الخطوة اللي فوق دي علطول هعرف symbol اسمه stack\_top ، هو ينفع أعمل كذا ؟ اه ما ده symbol ، ده في ال linker script كذا هيبقا اسمه symbol ، وده يعني address .

symbol is equivalent to address

---

خلي بالك إن ال symbol ده مش زي ال C ، يعني هو مش variable ، ال variable في ال C ليه مكان في ال memory وبيكون شايل جواه value .

---

يعني لو قولنا مثلاً  $int\ x$  فدي هتيجي مثلاً عند 0x10000 وهيبقا ال x جواها شايلة value ، لكن ال symbol بقا هو equal لل address ، يعني كانه هنا مثلاً equal لل 0x10000 ولكن لا يحمل جواه أي value .

---

فأنا بقوله :

stack\_top = . ;

يعني انت يا location counter بعد ما جمعت على نفسك 0x10000 اللي هي ال size بتاعت ال stack انت بقيت عند address ، هاتلي ال address ده automatic حطهولي في ال stack\_top .

---

وانت طبعاً سيد العارفين إن أنا همسك ال stack\_top دلوقتي وأجري على مين ؟ ال startup أحطله في المكان ده :

: reset

ldr sp, = stack\_top

bl main

---

ماينفعش في ال startup أعملها بالتخمين ، فأنا بخلي كل حاجه محسوبة automatic ، فال linker script بال command العجيب اللي اسمه location counter يقدر يرسم ال layout يقدر يحسب كل حاجه وبعد كذا عملت symbol اللي هو stack\_pointer خليفته equal ال dot عرفت أنا فين وهي دي اللي روحت حطيتها في ال startup بتاعي ، باصيتها لل sp ال stack pointer ، ما هو ده ال stack top بتاعي .

---

المنظر ده كويس بس فيه مشكلة ، ايه المشكلة ؟ عشان تعرف المشكلة تعالى ن compile ونفجر :

```
arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o startup.o -o learn-in-depth.elf $
```

```
arm-none-eabi-objdump.exe -h learn-in-depth.elf $
```

---

ال startup موجود في ال VMA عند ال 0x10000 وال text موجود عند 0x1000C وال data موجود عند 0x100e0 ، كدا مفيش مشكلة الدنيا كدا كويسة ، يعني هو بدأ من عند ال 0x10000 وقعد يجمع .

---

طب بشمهندس أنا مش مصدقك ، بس كدا متزعلش :

```
arm-none-eabi-objdump.exe -D learn-in-depth.elf $
```

---

تعالى نروح نشوف بقا ال reset/startup section في الحاجات اللي طلعت معنا ، عايزك تشوف المنظر تتمتع بيه كدا ، عند address 10000 فيه ال assembly code اللي بيقوله حطلي ال value دي ، طب انت جيب ال value دي منين يا بشمهندس ؟ من ال linker ، ال linker script حسبته وحطهولي من غير ما أنا أحسبه بإيدي ، بعد ما حط كل السكاشن وزود ال 0x10000 عرف انت واقف فين ، حطلي ال value وهي دي اللي هتتخط في ال stack pointer ، وشكلها طلع كدا :

```
10000 : 69632 #mov sp, #e3a0da11
```

---

يبقا ال mov instruction هتاخذ ال value دي وتخطها في ال stack pointer وده ال entry point بتاعتي عند ال address 10000 ، كدا ال processor أول ما يقوم يا جماعه هيشغل ال reset section ليه ؟ لأنك انت حطيت ال text بتاع ال startup.o في سكشن اسمه startup أو reset section مش هيفرق الاسم يعني .

---

موجود فين ال reset section ده ؟ موجود عند address 10000 ، يعني أول ال assembly code هيتنفذ في ال binary بتاعك هو ال startup ده وهو اللي هيحط فيه ال stack pointer لإنني محطوط عند ال entry point ، وهو ده اللعب كله .

---

تخيل بقا أنا ممكن أعمل مليون حاجه لو عندي في ال Interrupt vector table إن هو لما يجيله interrupt من timer يروح عند address 20000 هعمل نفس الحركة ، هروح أعرف سكشن لل vector وهربطه بسكشن جوا ال linker script أروح أحطه بال locator counter عند ال 20000 ، وأقدر أظبط إن فيه assembly أحطه بعينه عند address معين في ال Flash memory ، دا اللعب كله .

---

لو عملت ده تاني عشان نبص على حاجه بس :

```
arm-none-eabi-objdump.exe -h learn-in-depth.elf $
```

هتلاقى إن ال reset section اللي طالعة موجودة عند address 10000 ك VMA و LMA ، يعني ك burn-time و at runtime هما equivalent لبعض ما اتغيرتش .

---

## سؤال في المحاضرة

ال linker لما كتبناه خدنا ال stack\_top وروحنا كتبناه في ال startup ، طب هو ال linker بيشتغل في stage بعد ال startup أصلاً ، قال هيجيب ال value بتاعت ال stack\_top دي منين ؟

---

احنا في ال app.c كان بيروح في ال main ينده على Uart\_Send\_String طب هو فين ؟ مش موجود ، بس ايه اللي حصل ؟ طلع في ال symbol table إن فيه symbol اسمه Uart\_Send\_String وده unresolved ، مستنيين مين يعمله resolving ؟ ال linker .

---

نفس الكلام بقا ، ال symbols عامة حاجه واحدة ، يعني ايه symbol ؟ ال symbol يعني address ، فمثلاً ال Uart\_Send\_String ده هيطلعه symbol وال symbol ده equal ل address أنا مش عارفه لحد دلوقتي ، لما أجي بقا في ال linking وألاقي في حته تانية نفس ال symbol عند address عارفه بربط ال two symbols يعني بساويهم بنفس ال address ، هو ده معنى ال symbol .

---

ايه الفرق ما بين ال symbol وال variable ؟

ال variable هو عند address ولكن يقدر يشيل قيمة ، ال symbol ما يقدرش يشيل قيمة هو just an address .

---

فالموضوع بقا في ال startup هيبقا نفس الكلام ، ال stack\_top ده هيبقا symbol ويطلع في ال symbol table إنه ملهوش address يعني unresolved ، هيجي بعد كذا ال linker ياخد ال objects من بعض وهو عنده هو ال linker symbol فهيتحلله address من خلال ال location counter فبقا ي solve اسم ال symbol مع اسم ال symbol يساويه بال address ويخليه resolved .

---

ولو انت بقا شيلت اسم ال stack\_top ده من ال linker\_script وروحت عملت linking هيضرب منك ، لأن هيقولك unresolved symbol أنا مش عارف مين ال symbol ده

---

بعد الكلام ده بقا

## (Linker script (symbols

- symbol is the name of an address
- symbol declaration is not equivalent to variable declaration



- Each object has its own symbol table, the linker is resolving the symbols between all obj files
- Symbol also is used to specify Memory layout boundaries

---

ال boundaries في النقطة الرابعة دي هنستخدمها عشان نحجز مكان لل bss ون copy الداتا ودي هنشوفها في اللاب الثاني .

---

شكل ال symbol table بيكون فيه اسم ال symbol وبيكون فيه ال binding بتاع ال symbol هل هو معمول Global ولا Local ، وال Type بتاعه Data ولا Function ، وال Address بتاعه كام ، ولو فيه حاجه في ال Address معموله UNDEF فده معناه ان لسا ما اتعملهاش resolving وهيتعمل ليها resolving عن طريق ال linker .

---

ال relocation table دا بقا ان انت بتاخذ كذا object وبيكون كل واحد عنده ال symbol بتاعه وعنده ال address بتاعه .

---

## To read the symbols, you can use nm cross tool chain bin utility

ده مثال لو انت عايز تطلع بقا ال symbols وتتفرج عليها ، تعملها ازاى ؟ .. هنستخدم binary utility اسمها arm-none-eabi-nm.exe وتباصي ليها ال object بتاعك ، يعني أنا لو عملت كذا :

```
arm-none-eabi-nm.exe app.o $
```

---

دي هتطلع لي ال symbols اللي عندي في app.o ، هيطلعلك main وهتلاقي جنبها على الشمال T ، يعني ايه T ؟ يعني دي symbol موجود في ال text section ، وهيطلعلك string\_buffer وجنبه على الشمال D يعني ده symbol موجود في ال data section ، وهيطلعلك string\_buffer2 وده موجود في ال R اللي هي rodata section ، وهيطلعلك ال Uart\_Send\_String وجنبها على الشمال U يعني unresolved وهتلاقي ده الوحيد اللي مش طالعه address على الشمال على عكس اللي قبله .

---

لو جينا نشوف ال symbols بتاعت ال startup.o هنعمل كذا :

```
arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o $
```

دي عشان ن compile ، لإن احنا كنا نسينا نعمل compile لل startup ثاني بعد ما عدلنا فيه ، وبعد كذا هنعمل :

```
arm-none-eabi-nm.exe startup.o $
```

هيطلعلك السكاشن فيه ال main جنبها U يعني unresolved ، وال reset جنبها دي يعني ال symbol دا موجود في ال text section ، وال stack\_top هيطلع معايا برضو جنبه U لأنه هيكون Unresolved .

---

ولو جينا شيلنا بقا السطر ده من ال startup.s :

; . = stack\_top

لو جينا نعمل linking بعدها ، هيطلعلني :

'undefined reference to 'stack\_top

---

ال linker راح يدور على ال symbol ده ملقوش فمعرفش يطلع ال executable النهائي .

---

لو أنا بقا جيت عملت nm لل executable النهائي بعد ال linking ، بص بقا الحركة دي واستمتع ، أهم حاجة تستمتع :

arm-none-eabi-nm.exe learn-in-depth.elf \$

هتلاقي ال symbols كلها طلعت معاك ، وهتلاقي ال symbol reset اتحط عند ال 10000 ، وال stack\_top عند ال address بتاعه برضو ، هتلاقي كل حاجة اتحطت عند ال address بتاعها .

---

سؤال في المحاضرة :

طب يا بشمهندس هو فيه حاجة بتحدد ال size ما بين ال section وال section الثاني ؟ ال location counter ، هو مفيش حاجة ت limit ، يعني ما دام انت اصلاً بتحط كل السكاشن في ال Mem وال Mem عندها ال Length كبير فمفيش حاجة تعملك limit ، إلا لو انت عديتها .

---

يعني أنا لو عندي ال Length قليل أوي وال location counter عمال يحط ويخش على اللي بعده ، وهو بيخش على اللي بعده عدى ال memory اللي انت مسموحك ان ال section ده بيقا متعرف جواها ، ساعتها يضربلك error ، ال linker script يقولك خلي بالك انت طلعت برا ال boundaries بتاعت ال memory ، ودي هنشوفها اللاب الجاي ان شاء الله .

---

ان شاء الله المرة الجاية بقا ، احنا هنقعد نرسم ، عايزين نقعد نحسب ال data . من أوله لآخره ال start وال end بتاعه عشان هننقله ، نعمله copy من ال Flash لل RAM ، هنبدأ بقا نعرف symbols كتير وهنلعب لعب مقولنهاش هنا يعني .

---

خلاص احنا كدا حطينا ال stack\_top وميه ميه ، ناقصلنا ايه ؟ ناقص ان انا احط ال stack\_top في ال startup.s ودي حطيناها خلاص ، بعد كدا ناقص ايه بقا ؟ أعمل link ليهم كلهم مع بعض .. عملناها يا بشمهندس .

---

## Map File

فيه حاجة جميلة بقا اسمها ال map file ، ايه ال map file ده يا بشمهندس ؟ بيقولك ده يا جماعة بيوصفك كل المعلومات عن ال linker ، يعني انت وصفت ال linker وعملت السكاشن وحطيت ال addresses وعملت symbols ، هو بقا عايزك تطلع

المعلومات بعد ما عمل ال executable ده في file اسمه map وده فيه كل المعلومات اللي انت عملتها عشان تتأكد ان انت صح .

---

فلو انت عايز تعمل كدا ، بتيجي وانت بتعمل ال linking تقوله كالاتي :

```
arm-none-eabi-ld.exe -T linker_script.ld -Map=output.map app.o uart.o startup.o -o $  
learn-in-depth.elf
```

---

الحته دي اللي هي :

Map=output.map-

ده ببطلعلك map file وده فيه كل المعلومات اللي انت رسمت فيها ال layout بتاعت ال memory ، طب ده مفيد في ايه ؟ ..  
عشان تتأكد ان انت عامله صح .

---

فلما تيجي تفتحه هتلاقي فيه كل تعريفات السكاشن ، وهتلاقي فيه كل section بال symbols بتاعته .

---

## سؤال انترفيو

? Why .bss does not appear in the output section

يعني لو انت لاحظت هتلاقي ال bss مش موجود خالص ، ليه ؟ أولاً هو مش موجود خالص عشان أنا مش معرف حاجه uninitialized فدي أول نقطة .

---

النقطة الثانية بقا ، **حتى لو كان موجود المفروض مش هيبقا ليه إلا VMA ، مش هيكون ليه LMA** ، طب يا بشمهندس عايزين نجرب الكلام ده ، اللاب الجاي علطول في الحته دي .

---

كدا احنا معانا ال toolchain ، ال Code ، ال Linker script ، ال startup ، ومعانا البوردة ، ما تيجي بقا نستخدم ال executable اللي طالع ده نستخدم فيه حاجه اسمها strip أو objcopy عشان نطلع ال .hex ، ال .hex دي اللي احنا هنعمل ليها burn ون run .

---

عشان خاطر نطلع ال .hex بنقوله كدا :

```
arm-none-eabi-objcopy.exe -o binary learn-in-depth.elf learn-in-depth.bin $
```

---

احنا كدا طلعا ال binary ، وده بقا كله binary مفيهوش debug ، ال binary ده بقا هو اللي هروح أعمل ليه run دلوقتي .

---

من ضمن الحاجات الجميلة ان فيه command اسمه readelf بتديه ال elf image بتاعتك ، يجيبك معلومة ال entry point لو انت كنت عامل **ENTRY Command** وفيه ال section headers يعني كل سكشن عند address كام .

---

كدا ناقص ن run بقا .. ن run ازاى ؟

هنروح للمكان الي مسطيين فيه qemu ونقوله كدا :

```
qemu/qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn-in-depth.elf/.. $
```

ونكتب جنبه اسم ال machine اللي عايز أشغلها ، لأن فيه machines كتير ، وال -m دي بتقوله ال memory بتاعتك كام ، وال -M بتقوله اسم ال machine بتاعتك ايه ، والبوردة دي بيكون فيها display فأنا هقوله ما تشغليش ال graphic عشان الجهاز ما يضرش فهقوله -nographic ، وهقوله بعدها -kernel يعني ده المكان اللي أنا بياصي فيه ال executable بتاعي ، وبعد كدا هكتب ال executable بتاعي بقا اللي هو learn-in-depth.elf

---

لما تدوس enter بقا هتلاقيه طلعلك على ال Command Line :

```
learn-in-depth:keroles
```

وده اللي احنا كنا باعتينه بال uart ، ولا سحر ولا شعودة .

---

بشمهندس افرض انت ضاحك علينا وجاي عاملي حاجه اسمها qemu-system-arm وكاتب جواها printf للجملة دي ، أنا ايه اللي يضمنلي ان هو راح وبرمج وعمل الكلام ده ؟ .. يضمنك الكلام ده ان احنا لما نيجي نتعلم ال debugger هتقدر تخش على ال assembly وتحط break point وتشوف ال registers بتاعت ال processor وت debug كل حاجه كإنك على بوردة حقيقية ، هتمسك assembly assembly بس طبعاً مش وقته النهاردة .

---

احنا النهاردة المفروض اللي كنا هناخده كمان هو ال gdb لما تيجي ت debug ال software ، احنا كدا في slide 118 .

---

أنا طالب منكوا ايه بقا ؟

ال Lab الجاي اصعب بكثير من اللاب ده ، يعني اللاب ده أنا بس كدا بعلمكم الأساسيات ، اللاب الجاي نبدأ نخش على حاجه embedded بجد بقا ، نكتب startup كبير مش هيكون 4 سطور ، هنكتب startup ييجي 70 سطر مثلاً ، هنبدأ ن copy ال data section ، هنبدأ ن initialize ال bss ، هنبدأ نكتب linker script محترم معقد لو انت شوقته من بعيد هتقول لأ أنا مش عايز أقرأه ، بس انت لما تكتبه step by step وتبقا فاهم هتحس بمتعة غير عادية .

---

وخصوصاً بعد ما توصل لمرحلة إنك تفتح أي linker بقا وتشرحه لحد ، وتخش وتجادل وتعمل ، مفيش حاجه توقفك ان شاء الله .

---

اللي يهمني إن انت تعمل نفس الخطوات اللي أنا عملتها دي بإيدك ، يعني عايزك انت بإيدك تعمل analysis وال analysis اللي انت هتعملها دي على ال object وتطلعلي ال symbol table وتطلعلي السكاشن وال executable النهائي وتطلعلي السكاشن وال addresses تعملها في pdf أو word وترفعها على google drive وترفعلي صورة للبرنامج وهو شغال على ال .qemu

---

وبعد كذا تعملي ال startup وال linker وال learn-in-depth باسمك ، وال uart code والحاجات دي وترفعلي الحاجات دي على ال git repository .

---

لو انت عملت الحاجات دي بإيدك ، مشيت على المحاضرة واحدة واحدة ، قعدت تشتغل بإيدك وعملت كل حاجه ، أنا فرحان لإن اللاب اللي جاي أصعب بكثير ، فلما انت تعرف الأساسيات وتسمع مني اللاب الجاي وتعمل اللاب الجاي بإيدك ، ان شاء الله لما نيجي نخش في ال project اللي انتوا هتعملوه في الكورس ده ، فهو project هيمسك ال C يفحته وهيمسك ال embedded C يفحته .

---

فانت هتكتب بإيدك في ال project من غير ما تسمع مني ، أنا عايز اللي انت سامعه مني تكتبه دلوقتي وتمشي وتتعلم كل حاجه زي منا كنت ماشي دلوقتي ، عشان هيجي في وقت انت هتعمل وأنا مش هقولك أي hint ، هسيبك مع نفسك ، ابدع .

---

المره اللي جاية هنشغل على STM فهنشغل على proteus ومش هنستخدم أي IDE ، هنشغل kernel زي منا كنت شغال كذا ، مش هنستخدم لا STM32CubeIDE ولا Keil Uvision ، احنا هنكتب from scratch .

---

فاللي أنا طالبه منكم الأسبوع ده ، تمسكوا المحاضرة بتاعت النهارده تعملوها بإيديكم وتطلعولي ال assignment كالاتي على ال git repository بتاعك في Unit3\_lesson2 تعملي فيها :

app.c , uart.c , uart.h , startup.s , linker\_script.ld

بإيدك من غير ما تاخذ كود من أي حته ، بعد كذا ت run الكلام ده على qemu وتطلعلي ال screenshot وتحطها ، وبعد كذا تعمل pdf report ، خلي بالك ان الشركات فيه report بيتعمل ، تطلعلي فيه السكاشن بتاعت كل object وابه اللي كان في كل object معمول unresolved ، وبعد ما طلعت ال executable السكاشن اللي موجوده فيه بقا عند addresses كام ؟ ال entry point بتاعتك كام ؟ ال sizes بتاعت السكاشن كام ؟ ال stack\_top عند address كام ؟

---

تعمل report يعمل analysis لل binary اللي انت مطلعته وترفعهولي على جوجل درايف ، وتاخذ ال link بتاع جوجل درايف وال link بتاع ال repository تحطهولي ك comment للمحاضرة ، ده اللي أنا عايزه منك ، احنا كذا خلصنا المحاضرة الحمد لله .